Individuell rapport, TNM094

Isak Engström

Sällskapsspel för sammankopplade handhållna enheter

5 november 2020

Sammanfattning

Det blir allt vanligare med stora företag som utvecklar komplexa mjukvaruprodukter. Samtidigt arbetar små team tappert för att kunna mäta sig med de stora jättarna. Syftet med denna rapporten är att försöka skapa en plan för ett mindre team så att deras olika expertis kan frodas och skapa en produkt av värde.

Planen utefter skapandet av en produkt som är ett sällskapsspel för sammankopplade handhållna enheter. Ramverket Scrum används för att arbeta i iterativa sprintar, för att utveckla betydelsefulla inkrement av spelet under en 12 månader lång period. Utvecklarlaget består av 11 individer och rapporten utreder hur de ska arbeta på ett effektivt och agilt sätt.

Rapportens resultat är en plan över hur utvecklarteamet kan arbeta för att utveckla produkten. Den må behöva testas med ett verkligt team, men planen anses ändå vara passande att förhålla sig till.

Innehåll

Kapitel 1

Inledning

På den senaste tiden har mjukvaruutvecklingen skjutit i höjden. Det skapas allt fler, allt mer komplicerade program. Många utvecklarlag består av ett stort antal utvecklare, varav flera har olika kompetens och expertis. Trots detta finns det fortfarande lag som består av få antal utvecklare, som bland annat är vanligt vid start-ups och indie-utveckling. Alla utvecklare tacklar inte samma problem samtidigt, utan de delas upp för att bygga komponenter separat, som passar deras individuella kompetenser. Trots uppdelningen jobbar de mot ett gemensamt mål, att tillslut ha sammankopplat alla delar till en färdig produkt. Rapporten utreder den agila utvecklingsmetoden Scrum och hur den används för att optimera ett utvecklarlags arbete.

1.1 Syfte

Syftet med rapporten är att redogöra en planläggning av hur en grupp på 11 stycken utvecklare utformar ett sällskapsspel för sammankopplade handhållna enheter. Planen skall vara detaljerad och formad utefter ramverket Scrum. Projektutvecklingen sker i sprintar om två till fyra veckor. Strävan är att efter varje iterativ sprint ha landat i ett betydelsefullt inkrement av produkten och att utvecklingen sker på ett sätt som gör det möjligt att slutprodukten uppfyller kundens krav.

1.2 Frågeställning

Frågorna som ställs ämnar planeringen av projektet. Framförallt hur utvecklarlaget kan arbeta och kommunicera för att underlätta systemutvecklingen för varandra, kunden och andra intressenter. De frågor som förväntas besvaras i rapporten är:

- Hur kan appliceringen av Scrum se ut när ett lag på 11 personer utvecklar ett sällskapsspel för sammankopplade handhållna enheter?
- I vilken grad kan utvecklingen av ett sällskapsspel för sammankopplade handhållna enheter grenas upp och utvecklas parallellt för att stegvis sammanfogas till slutprodukten?

1.3 Avgränsningar

I rapporten görs ett flertal avgränsningar. Vid planeringen av projektet antas utvecklarlaget vara fulländat. Med det menas exempelvis att utvecklarna besitter den gemensamma kompetens som krävs för

att kunna utföra arbetet.

Exakt hur de iterativa implementationerna av produkten och hur den resulterande produkten ser ut redovisas inte i detalj. Anledningen är att produktidén kommer växa och utvecklas med tiden, bland annat beroende på kundens önskemål.

Kapitel 2

System och tekniska lösningar

Systemet som ska byggas i projektet är ett sällskapsspel vars spelplan placeras ut i den verkliga världen, men som endast syns genom skärmar på handhållna enheter. Spelarna delas in i två lag och skickar trupper längs fördefinerande vägar mot motståndarens bas. Målet är att försvara sin egen bas, samtidigt som motståndarens ska förstöras.

2.1 Grundläggande, initiala krav och systembegränsningar

För att främja förflyttningen av de handhållna enheterna ska spelplanen vara en viss storlek, minst 1 x 1 kvadratmeter. För att spelet ska räknas som ett sällskapsspel krävs ett flertal spelare. Det är lika många spelare i varje lag, det ska gå att spela en mot en, men även två mot två. Huruvida lagen ska bestå av fler spelare beror av spelplanens storlek och om det finns tillfredsställande uppgifter för alla spelare.

Spelet kommer av förklarliga skäl köras i realtid. Det ställer krav på hur snabb kommunikationen mellan de handhållna enheterna bör vara. Tidsfördröjningen bör inte vara mer än 200 millisekunder mellan två handhållna enheter. Om systemet skulle kommunicera via en server tillåts därför en latens på 100 millisekunder mellan server och handhållen enhet. Vidare besitter enheterna pekskärm och kamera, samt en skärm av minimala upplösningen 1920 x 1080 pixlar.

2.2 Målplattform

Spelet utvecklas för två olika operativsystem. Dessa är *Android* och *iOS*. Stöd för äldre versioner än *Android 8.1* och *iOS 11* implementeras nödvändigtvis inte. Systemet ska fungera för smartmobilerna *OnePlus 6* och *iPhone 7*, samt nyare modeller av *OnePlus* och *iPhone*.

De vanligaste smartmobilerna kör antingen Android eller iOS. Genom att kombinera utvecklingen för båda dessa operativsystem täcks stora delar av marknaden och sannolikheten att spelets användare har tillgång till sådana smartmobiler är stor. Trots de angivna avgränsningarna av operativsystemets version och smartmobilernas modell är sannolikheten stor att de kan köra spelet på båda framåt- och bakåt-kompatibla versioner av operativsystemen.

2.3 Grundläggande system-arkitektur

Systemet består av en mängd komponenter och element. Strävan är också att undvika att göra dessa till systemspecifika lösningar för de olika operativsystemen. På det sättet blir utvecklingen och prestandan mellan olika plattformar mer förutsägbar och lätthanterad för utvecklarna.

En huvudsaklig komponent är positionsavläsningen i det verkliga rummet. Spelet ställer krav på att användaren utför handlingar på smartmobilernas pekskärm med precision, därav vikten av en konsekvent avläsning. För denna precisions skull används datorseende-biblioteket *OpenCV* i kombination med *ArUco*-markörer. OpenCV är skapad i optimerad *C* och C++ och är ett bibliotek som designats för effektiva beräkningshastigheter, bland annat genom flerkärnsbehandling. Med tanke på den begränsade hårdvaran som finns i de relativt små handhållna enheter är detta en betydelsefull aspekt [1]. För att öka precisionen av smartmobilens position kan kalibrering av kamerans position göras i OpenCV med hjälp av ett antal funktioner som finns tillgängliga [2].

Spelets logik och beteende delas upp i skript som skrivs i *C#*, ett objektorienterat programmeringsspråk. Ett skript står för ett specificerat beteende istället för flera olika beteenden. På det sättet kan objekten i spelet tilldelas flera, mer generella skript, som i kombination ger objektet ett specifikt beteende.

En komponent som är avgörande för systemets arkitektur är de tredimensionella modellerna. Modellernas komplexitet begränsas utifrån smartmobilernas kapacitet. Modellerna har inte olika detaljnivå. Istället modelleras de i med en *Low Poly*-teknik för att vara hanterbara för enheterna när mängden objekt i scenen blir hög.

För att de handhållna enheterna ska kunna kommunicera med varandra i realtid används inte *Peer-to-peer-kommunikation* i första hand. Istället ska *Klient-Server-kommunikation* försöka implementeras. Förslagsvis används *UNET*, vars utveckling sker direkt i spelmotorn *Unity*. Om inte denna lösning bidrar med tillräckligt snabb kommunikation mellan enheterna, implementeras en alternativ serverlösning. Ett exempel på sådan är *UDP (User Datagram Protocol)*, som lämpar sig vid krav på låg latens.

2.4 Utvecklings-miljö

För att utveckla systemet för de båda operativsystemen används realtids-motorn Unity tillsammans med datorseende-biblioteket OpenCV. Motorn tillåter smidig multiplatform-utveckling medan biblioteket hanterar markörbaserad spårning via de handhållna enheternas kameror. Unity är smidigt, för systemet kan kompileras och köras direkt som en applikation på de handhållna enheterna. Unity erbjuder också möjligheten för att profilera GPU:n, CPU:n, minneshantering samt rendering. Profileringen är viktig för att kunna optimera spelet [3].

Systemets grundläggande arkitektur består av flera olika komponenter och moduler. Fördelen med Unity är det finns smidigt stöd för att implementera och använda många av dessa. Unity kan köras i flera olika miljöer, bland annat på maskiner som har *Windows* eller *Mac OS* som operativsystem. I Unity används C# som skriptspråk medan systemet kompileras i *C++* vid körtid.

Vid modellering ställs inga specifika krav på modelleringsprogram, detta på grund av att Unity kan importera flera olika filformat. En rekommendation är dock programmet *3Ds Max* och att objekten importeras till Unity i antingen OBJ-, fbx- eller 3ds-format.

Vid installation av Unity installeras även IDE:n *Visual Studio*. Både spelmotorn och IDE:n kan användas som debug-verktyg. Visual Studio erbjuder bland annat potentiella lösningar till fel som kan finnas i skripten. Systemet kompileras direkt i Unity som då skapar en applikation, antingen för Android eller

iOS.

2.5 Standarder

Spelet startas genom en installerad *.apk*-fil på Android-enheter. När spelet är färdigutvecklat kommer installationen ske via *Google Play*, en av Android-systemens onlinebutiker för applikationer som utvecklats av *Google*. För operativsystemet iOS installeras applikationen via *App store*, onlinebutiken för iOS-applikationer som utvecklats av *Apple*. När appen har installerats startas den via applikationen som är av *.ipa*-format. Dessa onlinebutiker erbjuder enkla metoder för att distribuera applikationen till de som vill använda den.

ArUco-markörerna kommer finnas tillgängliga i form av planscher som är av den önskade storleken. De kommer även finnas tillgängliga för nedladdning och utskrift via en hemsida för de som inte har tillgång till planscherna. Utskrifterna via hemsidan kan inte garantera den precision som planscherna gör, men det ger en vidare möjlighet för att spela spelet på andra platser.

Spelets moduler utvecklas och granskas var för sig om möjligt. Under projektets utveckling finns också en tydlig struktur över var filer ska placeras för att alla utvecklare ska hitta dem. På detta sätt kan konflikter och missöden undvikas i större grad. Om externa bibliotek ska användas i Unity kommer detta ske via statisk länkning. Det ger full kontroll över bibliotekens version, till skillnad från dynamisk länkning. Kontrollen innebär att spelet inte kommer sluta fungera på grund av oönskade uppdateringar av externa komponenter. För att generera referensdokumentation används *Doxygen*. Doxygen stödjer bland annat C++ och C# och det finns även plugins för att göra dokumentationen direkt från Unity.

Kapitel 3

Projekthantering

Den huvudsakliga teknik som styr samarbetet inom projektet följer principen av ramverket Scrum. Kapitlet reder bland annat ut hur den agila metoden ska anpassas för ett lag på 11 personer och om metoden kan behöva vissa justeringar för att anpassas till det specifika projektet. Här presenteras även utvecklingsstadier som är med vid för mjukvaruutveckling, utöver de som anges i Scrum.

3.1 Utvecklingsmetodik

Scrum erbjuder möjligheten att skapa en produkt som utgår ifrån en kunds önskemål. Metodens iterativitet leder till ett finslipat samarbete mellan systemutvecklarna, som ser till att produktens komponenter utvecklas i en prioriterad ordning.

Scrumteamet arbetar uppdelat för att skapa vissa komponenter åt gången och parallellt om så är möjligt. På grund av att teamet är så pass litet och alla inte besitter samma expertis inom alla områden, ställs stora krav på kommunikationen inom teamet [4]. Teamet arbetar även ständigt med återkoppling från systemtest, utförda både av Scrumteamet och utomstående systemtestare. Testen utförs för att minimera risken för buggar eller oönskad funktionalitet.

Sprintarnas längd är inte fixerade. Vid utvärdering och planering kommer Scrumteamet inse ungefär hur lång en kommande sprint behöver vara, men de bör ha en längd som är mellan två till fyra veckor. Målet med varje sprint är att implementera ett inkrement av produkten som är betydelsefullt. Under projektets gång kommer dessa inkrement vara olika komplicerade och därför kräva olika lång tid. Därav den lösa definitionen av sprintarnas längd.

Inte heller en sprints innehåll är fixerat. Vad som ska göras varje sprint finns med i *sprintbackloggen*. Innehållet kan antingen planeras om, eller så kan en sprint avbrytas helt och hållet [4].

3.2 Organisation

Uppdraget att utveckla produkten kommer från kunden. Kunden Utvecklarlaget har fått 12 månader på sig att färdigställa produkten, som är ett sällskapsspel för sammankopplade handhållna enheter. Kunden är involverad i utvecklingsprocessen för att säkerställa sig om att utvecklarlaget levererar vad som önskas.

Utvecklarlaget, eller det så kallade scrumteamet, består av 11 personer av olika erfarenheter inom systemutveckling. De delas upp i roller: produktägaren, scrummästaren och utvecklingsteamet. Av de nio personerna som finns med i utveklingsteamet hjälps alla åt med *QA (quality assurance)*, något

som blir allt mer viktigt ju längre in i projektutvecklingen de kommer. Vidare ansvarar de tillsammans för att utföra användartester [4].

Antalet personer i utvecklingsteamet gränsar till för många. Om fler teammedlemmar skulle visas behövas, skulle teamet eventuellt kunna delas upp i två scrumteam, som jobbar mot samma sprintmål [4].

Hur ett scrumteam väljer att fördela arbetet sker via självorganisering. Däremot bör utvecklarlaget besitta tvärfunktionalitet som innefattar ett antal kompetenser [4]. Två av dem besitter kunskap för att länka de handhållna enheterna. Två andra är duktiga frontend-utvecklare, som kan tänkas arbeta med animering, modellering, texturering och ljud. Tre utvecklare kan hantera AR och skriptning, så att de exempelvis skulle kunna arbeta med Unity, OpenCV och ArUco för att kunna koppla samman alla olika komponenter till en produkt. De två sista personerna innehar skicklighet inom både back- och frontend, vilket lämpar sig för att anpassa produkten utefter varsitt operativsystem. En av dem skulle därför kunna anpassa produkten till Android medan den andra adapterar spelet till iOS.

Genom att ha ett utvecklarlag som består av minst två personer som besitter liknande kompetenser innebär att det ökar sannolikheten att det alltid finns en person av varje kompetens på plats. Vare sig anledningen att någon frånvarar beror av semestertid eller sjukdagar.

Produktägaren ser till att alla vet vad de ska prioritera. Denna teammedlem ansvarar även för produktbackloggen, sköter den huvudsakliga kommunikationen med kunden, samt bestämmer när användartester ska utföras. Trots att ägaren ansvarar för produktbackloggen utför denne inte arbete *från* denna. Tvärfunktionaliteten som angivits i rapporten är bara till för att tydliggöra de kvalifikationerna som teamet bör äga, hur den egentliga fördelningen sker är helt upp till produktägaren [5].

Scrummästaren är till hjälp för både utvecklingsteamet och produktägaren. Individen ser till att alla uppför sig utefter ramverkets principer och regler. De resterande medlemmarna i utvecklarlaget får hjälp av mästaren att effektivisera interaktion internt och externt[4]. Inte heller mästaren utför arbete från produktbackloggen.

Slutligen finns de utomstående systemtestarna. De individerna som inte tillhör utvecklarlaget men ändå arbetar med QA. De är med och utför *grey-box-* och *black-box-test*, där testaren delvis eller inte alls vet om produktens bakomliggande struktur [6].

3.3 Tidsplan

Utvecklarlaget har 12 månader på sig att skapa en färdig produkt som uppfyller kundens krav. Från start till slut genomförs sprintar och varje sprints längd fastställs utav utvecklarlaget. Det kan ske max 24 stycken sprintar och minst 12 stycken om sprintarnas längd hålls mellan två och fyra veckor långa. Om sprintarna hålls kortare än så finns det en viss risk att det inte hinner ske ett betydelsefullt framsteg i produktutvecklingen. Sprintar längre än fyra veckor kan dock leda till brist i kommunikation och att utvecklarna gräver in sig i metoder under längre tid, som sedan inte visar sig vara ideala för produkten. Viktigt vid planeringen av sprintarnas längd är att de ska gå jämnt ut när de 12 månaderna är över.

Varje sprint består delvis av de tre beståndsdelarna: *Sprintplanering*, *Sprintgranskning* och *Sprintåterblick*. De har en tidsgräns som varierar beroende på beståndsdel samt hur lång sprinten som ska utföras är.

Sprintplaneringen utförs i början av varje sprint. För fyra veckor långa sprintar är maximala tidsgränsen åtta timmar. Teamet kan tänka att det ska avsättas två timmars planering för varje vecka som sprinten består av. Under planeringen fastställs *Sprintmålet*, det som ska uppnås under sprinten. Det nås genom att följa produktbackloggen och ger utvecklarna ett tydligt syfte att jobba emot under sprinten.

Sprintgranskningen utförs när sprinten närmar sig sitt slut. Tidsgränsen är fyra timmar för en fyra veckor lång sprint. Laget kan tänka sig en timmas granskning för varje vecka som sprinten består av. Här sker en granskning av sprinten som börjar ta slut och produktbackloggen konsulteras och får eventuella ändringar för att optimera inkrementets kaliber.

Sprintåterblicken äger rum efter granskningen av en sprint och innan nästa sprintplanering. En fyra veckor lång sprint begränsar återblickens längd till tre timmar. Lägre gräns kan tänkas sättas för kortare sprintar. Under sprintåterblicken sker en tillbakablick över sprinten som varit och förbättringar till nästa sprint diskuteras [4].

Den iterativa processen avgör exakt när vissa komponenter ska vara färdigutvecklade. Däremot är komponenterna för att koppla samman enheterna med varandra över nätverk viktig för att koppla samman resterande komponenter. Design av spelplanen är också viktig att få klar tidigt. Inte i detaljnivå, utan fokus bör ligga på att placera ut hinder och liknande. Andra komponenter är dock inte beroende av hindrens modell eller utseende, vilket betyder att sådant inte behöver prioriteras i produktbackloggen. Utvecklingen av markörspårningen kan ske parallellt med utvecklingen av resterande komponenter. Det bör dock fastställas tidigt, helst i samband med utvecklingen av nätverket, för att fastställa om den valda metoden för spårningen är lämplig.

3.4 Milestones och leverabler

Utöver sprintmålen och att produkten ska vara klar efter 12 månader finns tre ytterligare utvecklingsstadier som delar in sprintarna i olika områden. De kallas *pre-alfa-*, *alfa-* och *beta-stadierna*. Exakt när dessa äger rum beror på sprintarnas längd och vad utvecklarlaget kommer överens om tillsammans med sin kund. Att sträva efter är dock att pre-alfa sker sex månader in och beta äger rum när nio månader har gått, med alfa ungefär mittemellan de båda stadierna. Vid dessa tre milstolpar är det viktigt att QA utförs, både av utvecklarlaget och utomstående systemtestare. Kunden ska också ha möjlighet att pröva spelet.

Vid pre-alfa bör all funktionalitet finnas tillgänglig, även om systemversionen är tillåten att vara full av buggar. Det innebär att nätverket och rörelsespårningen är implementerade, tillsammans med den grundläggande spellogiken. Det skulle kunna innebära att det går att dela in sig i lag och att ett av dem kan vinna. Vid detta stadie sker grey-box-, black-box- och *white-box-testning*, där den sistnämnda metoden innebär att inget bakomliggande är dolt. Om utomstående får tillåtelse att utföra white-box-test är upp till kunden, som i grund och botten bestämmer vem som har tillgång till exempelvis källkod. Kunden har alltid rätten att ändra sig när det kommer till produkten, i detta stadie bör utvecklarlaget vara beredda på större sådana.

Vid alfa-steget sker ytterligare tester och eventuella förändringar. Här bör spellogiken och modelleringen vara nästintill klara. Kunden är relativt nöjd med spelets utformning men kan önska att mindre komponenter och funktioner läggs till eller att andra finslipas.

Till slut är det dags för beta-fasen. Här är produkten nästan färdigutvecklad. Inga nya komponenter läggs till, inte heller ny funktionalitet eller spellogik. Kvar återstår arbete med att lösa buggar samt optimering av produkten, samt eventuell modellering för att förfina spelplanen. Här kan produkten eventuellt läggas upp på Google Play och/eller App store. Om det finns tid bör också kvarstående dokumentation göras här. Detta på grund av att vidareutveckling av produkten kan ske långt in i framtiden och av andra utvecklare.

Den sista milstolpen är den färdiga produkten. Följs de tidigare milstolparna har behovet av *crunch time* minskat inför denna milstolpe. Om produkten inte redan gjordes tillgänglig i onlinebutikerna under beta-fasen är det dags att göra det nu. Huruvida vidareutveckling av produkten ska ske beror på kunden och utvecklarna och ligger utanför ramen av 12 månader som anslagits.

Kapitel 4

Rutiner och principer

Hur utvecklarlaget och andra intressenter arbetar med varandra påverkar utveckling och slutresultat hos produkten. Det påverkar även det psykiska välmåendet hos de inblandade. Transparens och kommunikation är en nyckel till ett välfungerande agilt arbete.

4.1 Mötesprinciper och rutiner

Det kan vara lämpligt att informera kunden om den agila metoden som laget följer och när det passar bäst att hålla möte och liknande. Trots detta är det viktigt att komma ihåg att utvecklarlaget arbetar åt kunden och ska anpassa sig efter dennes/dess önskemål.

Kommunikationen med kunden sker delvis i form av kundmöten. Dessa sker helst efter varje sprint. De bör inte ske oftare än så, eftersom produkten inte är leverabel under sprintens gång. Om kundmötena sker mer sällan kan det visa sig att produkten tar en helt annan riktning än kunden egentligen önskar.

Utanför sprintarna sker det *dagliga Scrummötet*. Det är ett möte som sker i början av varje arbetsdag mellan parterna i utvecklarlaget. Under mötet tas det viktigaste upp som hänt sedan det senaste dagliga Scrummötet som utvärderas, följt av planering av dagen prioriteringar. Mötet är begränsat till 15 minuter [4]. Detta möte ser till att alla vet vad de ska göra, samt att de får reda på vad de andra utvecklar.

Utöver dessa möten sker också Sprintplaneringen, Sprintgranskning och Sprintåterblick som beskrevs närmare i kapitel 3.3 av rapporten. Gemensamt för scrummöten är att de har en tidsgräns som inte överskrids. Mötena bör inte heller vara mycket kortare än den avsedda tiden. Den lägre tidsramen är viktig för att se till att teamet tar del av varandras arbete, medan den högre gränsen hindrar mötena från att ha för lågt tempo.

4.2 Kravhantering och -spårning

Kravhanteringen sker dynamiskt via produktbackloggen. Produktägaren ser till att hålla den levande utifrån de krav som ställs av kunden och vad som anses vara de mest värdefulla prioriteringarna inför kommande sprintar. Även om produktägaren har det huvudsakliga ansvaret får denne hjälp av Scrummästaren att anpassa loggen så den följer Scrum.

Vissa krav som ställs i produktbackloggen riktas inte mot alla i utvecklarlaget. Det kan exempelvis finnas en specifik animering som kunden värderar högt. Det kan också vara ett krav som påverkar ett

flertal utvecklare. Ett exempel är om kunden helt plötsligt vill rikta produkten mot en yngre målgrupp. Det kan betyda att modeller och spelmekanik behöver ändras.

QA utförs av alla i Utvecklingsteamet, samt av utomstående systemtestare. Åsikter om produktens kvalité bokförs flitigt för att sedan presenteras för utvecklarlaget och kunden.

4.3 Versionshantering, -system och rutiner

Utvecklarlaget använder *GitHub* för versionshantering. Utvecklandet av produkten sker i olika *branches*. Det finns en *masterbranch* som alla färdiga komponenter läggs till i. Medan en utvecklare arbetar med en komponent så sker det i en separat och egen branch. Detta för att undvika ofta förekommande konflikter mellan utvecklare i masterbranchen. För att undvika konflikter som kan tillkomma om flera utvecklare jobbar i samma scen, görs först en kopia av den scen som utvecklaren ska arbeta i. Därefter förs detta in i den önskade scenen så att alla får tillgång till scenens hierarki. Många filer som finns i ett Unity-projekt ska måste inte laddas upp till GitHub. Dessa filers direktiv går att ange i en speciell fil som gör att de ignoreras.

För att göra en effektiv versionshantering sker ofta *commits*. Dessa gör utvecklaren genom att först utveckla en mindre komponent eller funktionalitet och lägga till en kommentar på vad som lagts till. Genom frekventa commits kan utvecklarlaget se tydligt när saker lagts till och gå tillbaka till dessa ifall det skulle behövas.

4.4 Arkitektur- och programdesign, standarder och rutiner

Systemarkitekturen för hela projektet är inte väldefinerad vid användandet av agila metoder som Scrum. Det kan skapas en initial arkitektur som underlättar vid starten av utvecklingen eftersom att det sker flera möten med kund och internt i utvecklarlaget innan den första sprinten väl sätter igång. Denna arkitektur liknar designprincipen *RDUF (Rough Design Up Front)*.

På grund av den begränsade framförhållningen som följer av den agila metodiken som används när väl sprintarna sätts igång, kan systemets arkitektur behöva uppdateras innan nästa sprint. Detta görs under sprintgranskningen. Fördelaktigt är att detta sker en tid innan nästkommande sprint. Då hinner sprintbackloggen planeras och kommer vara redo när teamet ska sätta igång med nästa sprint.

Utifrån arkitekturen och sprintbackloggen växer sedan programdesignen fram i form av mål av leverabla komponenter.

4.5 Dokumentationsprinciper och rutiner

Doxygen används via Unity för att spara dokumentationen av koden. För utvecklarlagets skull sker noga dokumentation i källkoden för alla olika metoder. Dokumentation sker även i header-filer för att underlätta för andra intressenter som antingen ska utföra grey-box-testning eller som kan tänkas vilja använda koden i framtiden för att vidareutveckla projektet, men här behöver inte privata metoder listas och beskrivas. Kunden förstår förhoppningsvis vikten i att projekten blir dokumenteras väl, ifall denne skulle vilja utföra vidareutveckling av projektet, antingen med samma utvecklarlag eller ett helt annat team. Den referensmanual som genereras av Doxygen läggs i GitHub så att den finns tillgänglig vid programbyggnings-processen.

Vid varje möte ska en på plats utsedd sekreterare föra protokoll, för att minimera risken att viktiga

beslut och idéer glöms bort. Det lämpar sig även för vardera part i utvecklarlaget att föra personliga anteckningar eftersom att sekreteraren kan råka missa eller inte alltid förstå vissa detaljer.

4.6 Kvalitetssäkring

QA genomsyrar hela utvecklingsprocessen av projektet, från start till slut. Utvecklarna är indelade i kompetensområden bestående av minst två utvecklare. Under varje sprint utför den en av utvecklarna QA inom det egna kompetensområdet, medan en annan utvecklare utför QA på ett annat kompetensområde. Efter varje sprint roterar de mellan det egna områden och något annat. På det sättet kan de kvalitetssäkra att alla komponenter med alla olika kompetenser som teamet besitter.

Kvalitetstest utförs även av kunden och andra intressenter, via black-box- (körtester) och grey-box-tester (körtester och viss kodgranskning). Intressenter är de som frivilligt vill utföra systemtester, antingen efter att ha anmält sig eller tillfrågats. Om kunden eller intressenterna ska utföra QA i form av kodgranskning behöver de ha erfarenhet av hur ett liknande projekt kan byggas. Total kodgranskning, det vill säga white-box-tester får utföras av utvecklarteamet eller av kunden och de intressenter som fått kundens tillåtelse. Desto större tillgång intressenterna får, desto större sannolikhet är det att de kan upptäcka buggar eller optimeringsmöjligheter.

Körtesterna för intressenterna kan vara antingen guidade eller helt fria. På det sättet kan olika sorters problem upptäckas.

Komponenter och metoder implementeras och testas en i taget, för att sedan lättare kunna lokalisera buggar som uppstår. För om spelet slutar fungera efter att en metod har implementerats, ligger buggen med stor sannolikhet i den metoden.

Kapitel 5

Analys och diskussion

Efter att ha genomfört en teoretisk planering av projektet finns det en del att analysera och diskutera. De val av metodik och projekt-utförande som gjorts är baserade på fakta och om det har funnits en osäkerhet i metodval har även alternativa lösningar föreslagits. Det lämpar sig ändå att diskutera huruvida planen kan vara en realistisk grund till utförandet av de mål som ställts i projektet.

5.1 Resultat

Värt att diskutera är antalet utvecklar som projektet planerats för. Till en början bestod utvecklarlaget av sex stycken personer, men under rapportens gång ökade antalet för att tillslut landa på 11 individer. Det innebär att utvecklingsteamet består av max antalet utvecklare som rekommenderas. Om teamet hade bestått av fler individer så hade samordningen enligt den agila metoden visats vara komplex. Samtidigt hade ett större team underlättat, eftersom det exempelvis skulle behövas dedikerade QA-testare som besitter olika kompetens för att säkerställa att produkten lever upp till kraven. Alternativt hade det funnits ytterligare ett utvecklarlag, för att då få en förbättrad samordning trots att det totalt sett finns fler utvecklare [4].

Ett team på 11 personer är fördelaktigt när det kommer till planerad eller oplanerad frånvaro av någon teammedlem. Om de som besitter samma kompetens undviker att ha semestertid samtidigt kan en av dem hela tiden vara med och utveckla. De kan då vara där och lösa fixa något som plötsligt läggs till i sprintbackloggen. I ett mindre team hade de eventuellt behöva ta semester samtidigt, eller hoppas på att en av medlemmarna med unik kompetens inte behövs under dess semester.

Även om planen är detaljerad är det ändå värt att nämna att den är preliminär. Det kan visa sig att någon vital del av systemet inte fungerar och hela kapitel 2 därför måste göras om. Fördelen med detta projekt är dock att komponenterna tål att utvecklas parallellt till en hög grad. Så länge nätverks-komponenten implementeras går det att utveckla resterande komponenter helt separat. Därav blir de inte känsliga för eventuella förseningar av andra leverabla komponenter.

Det som presenteras i kapitel 3 och kapitel 4 anses däremot utgöra en grund för hur projektgruppen ska sträva efter att arbeta, dels för att må bra, men även för att uppnå de krav som ställs på dem angående projektutvecklingen.

5.2 Arbetet i ett vidare sammanhang

När produkter ska utvecklas och förväntas vara färdiga vid en viss deadline blir arbetet lätt stressfullt. Det kan uppstå stressiga situationer även om den agila metoden i kombination med de angivna

milstolparna strävar efter att minimera detta när leveransen av den slutgiltiga produkten närmar sig. Crunch time kan förekomma, inte bara i slutet utan det kan uppstå när vissa sprintar närmar sig sitt slut.

Om inte arbetet sker lika parallellt som förväntat kan hela produktionen stå still om ett planerat inkrement inte levereras. Det kan exempevis bero på att utvecklare har fått för avancerade uppgifter. Om inte situationen behandlas på ett godtyckligt sätt kan konflikter uppstå mellan utvecklarna. Även om tid och resurser inte utnyttjas till fullo är det viktigt att teammedlemmarna mår bra. Därför kan det vara värt att ändra sprintbackloggen eller helt avbryta en sprint.

Kapitel 6

Slutsatser

Som nämnts i inledningen ämnar rapporten utreda hur ett mindre utvecklarlags arbete kan optimeras. Utifrån de angivna förslagen bör de kunna utföra arbetet på ett hållbart sätt, både för teamets välmående och slutproduktens resultat.

Rapporten redogör ett förslag på en detaljerad plan över hur ett team på 11 personer kan utveckla ett sällskapsspel för sammankopplade handhållna enheter. Detta genom en iterativ och parallell systemutveckling som följer den agila metoden Scrum. Värt att nämna är att planen är teoretisk. För att säkerställa sig över om planen verkligen fungerar skulle den behöva appliceras i ett praktiskt projekt av samma kaliber. Om det är möjligt praktiskt skulle rapporten därefter kunna användas som en guide för liknande projektplaner.

Litteraturförteckning

[1] OpenCV Dev Team, *About*, OpenCV, u.å. hämtad: 2019-02-16
https://opencv.org/about.html

[2] OpenCV Dev Team, *Theory*, OpenCV, 2019-02-21, hämtad: 2019-02-21
https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

[3] Unity Technologies, *Profiler overview*, 2019-04-15, hämtad: 2019-05-03
https://docs.unity3d.com/Manual/Profiler.html

[4] Jeff Sutherland och Ken Schwaber, *Scrumguiden*, 2013-07-01, hämtad: 2019-02-05
https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-SE.pdf

[5] Scrum Alliance, *Learn About Scrum*, u.å. hämtad: 2019-02-05
https://www.scrumalliance.org/learn-about-scrum

[6] Software Testing Fundamentals, *Black Box Testing*, u.å. hämtad: 2019-02-14
http://softwaretestingfundamentals.com/black-box-testing/