Pasi Kähönen

Scrum – sovelluskehitysprosessi pelikehityksessä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

24.9.2013

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Pasi Kähönen Scrum – sovelluskehitysprosessi pelikehityksessä 44 sivua + 1 liitettä 24.9.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Miikka Mäki-Uuro Lehtori Jorma Räty

Tässä opinnäytetyössä käsitellään Scrum-sovelluskehitysprosessin soveltuvuutta pelinkehityksessä. Raportissa selvitetään, mitä Scrum-sovelluskehitysprosessi pitää sisällään sekä mitä työkaluja ja tekniikoita käytettiin projektin toteuttamiseen. Projektilla ei ollut tilaajaa, vaan työ oli suunniteltu näytetyöksi omaan portfolioon.

Projektissa sovellettiin ketteriin sovelluskehitysmenetelmiin kuuluvaa Scrum-projektinhallintamenetelmää. Raportissa esitellään ensin Scrumin synty sekä toiminta teoriassa. Tämä pitää sisällään toimintaperiaatteiden läpikäynnin, scrum-tiimin jäsenten roolit, palaverit, scrumin tuotokset sekä ”valmiin” määrittelemisen.

Teoriaosuuden jälkeen esitellään peliprojekti Knights Quest, joka toteutettiin Scrum-sovelluskehitysprosessia soveltamalla. Osiossa käydään läpi pelin idea, kuinka se sai alkunsa, sekä selvitetään, kuinka Scrum-mallia on sovellettu projektin varrella.

Scrumin käyttö toi projektiin selkeyttä ja joustavuutta vaihtelevasta aikataulusta ja pienestä kahden hengen kehitystiimistä huolimatta. Menetelmästä jäi mieleen kuva toimivasta kokonaisuudesta, joka soveltuu ehdottomasti myös pelikehitykseen.

Avainsanat	Scrum, scrumtiimi, kehitystiimi, tuoteomistaja

Author(s) Title Number of Pages Date	Pasi Kähönen Scrum – development framework in game development 44 pages + 1 appendices 24 September 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Miikka Mäki-Uuro, Senior Lecturer Jorma Räty, Senior Lecturer

This study discusses the suitability of Scrum development framework in game development. The report explains what Scrum development framework consists of and what tools and techniques were used during the project.
Scrum methodologies were slightly altered during the project to fit the needs of the team. The report starts with explaining what Scrum is, how it came to be and how it works in theory. This part also includes theory about the operating principles, the roles of the members in a Scrum team, the output of Scrum and the definition of 'done'.

The second part presents a game called Knights Quest, which was implemented by using applied theories of Scrum development framework. This part of the report consists of explaining the game idea and how it was born and an explanation on how the Scrum model had been applied to the project along the way.

The use of Scrum on the project brought clarity and flexibility to it, even though the schedule varied a lot and the team consisted only of 2 persons. The overall image form the Scrum development framework was that it does works and is definitely a suitable method in game development.

Keywords	Scrum, scrumteam, scrum development team, product owner

Sisällys

Lyhenteet

OOPSLA ’95. Object-Oriented Programming, Systems, Languages & Applications.Olio-ohjelmoinnin, järjestelmien, kielten & sovellusten vuoden 1995 konferenssi.

JPA. Java Persistence API. Teknologia, jonka avulla oliot voidaan tallentaa relaatiotietokantaan.

LWJGL. Lightweight Java Game Library. Avoimen lähdekoodin kirjasto tietokonepelien kehittäjille.

OpenGL. Open Graphics Library. Avoimen lähdekoodin ohjelmointirajapinta, joka on tarkoitettu graafiseen ohjelmointiin.

OpenAL. Open Audio Library. Avoimen lähdekoodin ohjelmointirajapinta, joka on tarkoitettu ääniohjelmointiin.

OpenCL. Open Computing Language. Avoimen lähdekoodin ohjelmointirajapinta, joka on rinnakkaisohjelmointiin tarkoitettu viitekehys.

XML. Extensible Markup Language. On rakenteellinen kuvauskieli, joka auttaa jäsentämään laajoja tietomassoja selkeämmin.

UML. Unified Modeling Language. Ohjelmistojen määrittelyyn, suunnitteluun ja dokumentointiin tarkoitettu graafinen kieli.

1 Johdanto

Tämä raportti käsittelee Scrum-sovelluskehitysprosessin soveltamista pelinkehityksessä. Työn tarkoituksena oli selvittää, kuinka Scrum-sovelluskehitysprosessi toimii ja pyrkiä toteuttamaan tietokonepeli tätä sovelluskehysmallia hyödyntäen. Pelialalla korostuu ennen kaikkea oman osaamisen näyttö. Tästä syystä tekniikat oli valittu sopivan haastaviksi tiimin osaamisen kehittämistä ajatellen.

Tutkimusongelmana oli selvittää Scrum-sovelluskehitysprosessin käsitteitä sekä teoriassa että käytännössä. Suurimmaksi haasteeksi muodostui soveltaa malli kehitystiimimme kokoon ja aikatauluun sopivaksi. Keskeisiä kysymyksiä olivat:

- Miten Scrum toimii teoriassa?
- Miten sovellamme Scrumia toimivasti projektissamme?
- Toimiiko Scrum pelinkehityksessä?

Käsittelyssä oli pelin toteutus Scrumilla, peliin liittyvä tekniikka sekä peli-idean synty. Aihepiiri piti sisällään mm. Scrum-viitekehyksen eri vaiheet, käytettävät rajapinnat ja tekniikat sekä toimivan koodin esittelyä testien välityksellä. Projektissa saavutettiin ne tavoitteet, joita itselle ja projektikokonaisuudelle oli määritelty päättötyön sallimassa aikarajassa.

2 Scrum

2.1 Mikä on Scrum ja ketterä ohjelmistonkehitys?

Scrum on projektinhallintaan tarkoitettu viitekehys, joka on yleisesti käytössä ketterässä ohjelmistonkehityksessä, se on myös sovellettavissa yleisesti projektinhallinnassa. Scrumilla sekä muilla ketterillä menetelmillä pyritään minimoimaan riskejä jakamalla ohjelmistonkehitys lyhyisiin iteraatioihin, joiden tyypillinen kesto on yhdestä neljään viikkoa. Iteraatiot ovat ikään kuin pieniä ohjelmistoprojekteja ja sisältävät kaikki tarvittavat tehtävät uusien toimintojen julkaisemiseen. Ketterissä ohjelmistonkehitysprojekteissa pyritään julkaisukelpoiseen ohjelmistoon jokaisen iteraation lopussa, vaikka toiminnallisuus ei olisi suuresti kasvanut edelliseen verrattuna. Iteraation lopussa projektin prioriteetit arvioidaan uudelleen ja päätetään seuraavan iteraation sisältö.

Ketterissä menetelmissä suora viestintä on tärkeämmässä roolissa kuin kirjoitettu dokumentaatio. Tiimit työskentelevät usein samassa työtilassa ja koostuvat vähintään ohjelmoijista sekä heidän asiakkaistaan. Tiimin jäseninä voi myös olla testaajia, käyttöliittymäsuunnittelijoita, teknisiä kirjoittajia sekä päälliköitä.

Ketterissä menetelmissä korostetaan toimivan ohjelmiston olevan ensisijainen edistyksen mittari, eikä pitkälti valmiilla dokumentaatiolla ole niin suurta merkitystä kuin useimmissa perinteisemmissä malleissa. Tästä huolimatta, toisin kuin voisi luulla, suunnittelu on tärkeässä roolissa ja sitä tapahtuu koko projektin ajan. Tämä myös mahdollistaa sen, että suunnitelmien muuttaminen tarpeen mukaan on helpompaa ja edullisempaa, kuin perinteisissä malleissa.

Vuonna 2001 julkaistiin ns. Agile Manifesto (”Ketterä manifesti”) 17 merkittävän ketterän kehityksen puolestapuhujan toimesta. Manifestin sisältö on seuraava:

> ”Löydämme parempia tapoja tehdä ohjelmistokehitystä, kun teemme sitä itse ja autamme muita siinä. Kokemuksemme perusteella arvostamme:
>
> **Yksilöitä ja kanssakäymistä** enemmän kuin menetelmiä ja työkaluja
>
> **Toimivaa ohjelmistoa** enemmän kuin kattavaa dokumentaatiota
>
> **Asiakasyhteistyötä** enemmän kuin sopimusneuvotteluita
>
> **Vastaamista muutokseen** enemmän kuin pitäytymistä suunnitelmassa.
>
> Jälkimmäisilläkin asioilla on arvoa, mutta arvostamme ensiksi mainittuja enemmän.”

Manifesti muovautui kokouksessa, jossa oli tarkoitus luoda yhteinen pohja ketterille menetelmille ja edistää täten ketterän ajattelun leviämistä. [1;2;3;4.]

2.2 Scrumin lyhyt historia

Scrum määriteltiin ensimmäistä kertaa vuonna 1986 Japanilaisten Hirotaka Takeuchin ja Ikujiro Nonakan toimesta artikkelissa ”New New Product Development Game” joustavaksi, holistiseksi tuotteen suunnittelustrategiaksi, missä tuotantoryhmä pyrkii toimimaan yhtenä yksikkönä yhteisen tavoitteen saavuttamiseksi. He kuvailivat uutta lähestymistapaa kaupallisten tuotteiden kehitykseen, joka nostaisi kehityksen nopeutta ja joustavuutta. He kutsuivat lähestymistapaa holistiseksi tai rugbyksi, sillä koko projekti toteutetaan yhdellä monitaitoisella ryhmällä usean keskenään lomittuvan vaiheen läpi, missä ryhmä ikään kuin pyrkii pääsemään yhdessä maaliin syötellen palloa vuoronperää toisilleen.[5.]

Rugbyssä scrum viittaa pelin uudelleen käynnistämiseen pienen rikkeen jälkeen. 1990-luvun alussa, Ken Schwaber käytti yrityksessään Advanced Development Methods kehitysmallia, josta Scrum lopulta muovautuisi. Samoihin aikoihin Jeff Sutherland, John Scumniotales ja Jeff McKenna kehittelivät vastaavaa lähestymistapaa Easel Corporationilla, ja he olivatkin itse asiassa ensimmäiset jotka viittasivat kehitysmalliin sanalla Scrum.

1995 Sutherland ja Schwaber esittelivät yhdessä dokumentin Austin Texasissa pidetyssä OOPSLA ’95 (Object-Oriented Programming, Systems, Languages & Applications ’95)-tapahtumassa, jossa kuvailtiin Scrum-menetelmäoppia. Seuraavina vuosina Schwaber ja Sutherland tekivät yhteistyötä muovatakseen menetelmäopista, omien kokemuksiensa pohjalta ja alan parhaista tavoista kokonaisuuden, joka nykyään tunnetaan nimellä Scrum.

2001 Schwaber työskenteli Mike Beedlen kanssa esitelläkseen tavan kirjassa ”Agile Software Development with Scrum”. Sen lähestymistapa projektin suunnitteluun ja hallintaan oli tuoda mukaan päättävässä asemassa oleva yksilö toiminnan ominaisuus ja varmuus tasolle.

Vaikka sana Scrum ei ole lyhenne, monet yritykset käyttävät siitä kirjoitusmuotoa ”SCRUM”. Tämän epäillään johtuvan siitä, että eräässä aikaisessa Ken Schwaberin määritelmässä otsikko oli kirjoitettu kapiteelikirjaimin.

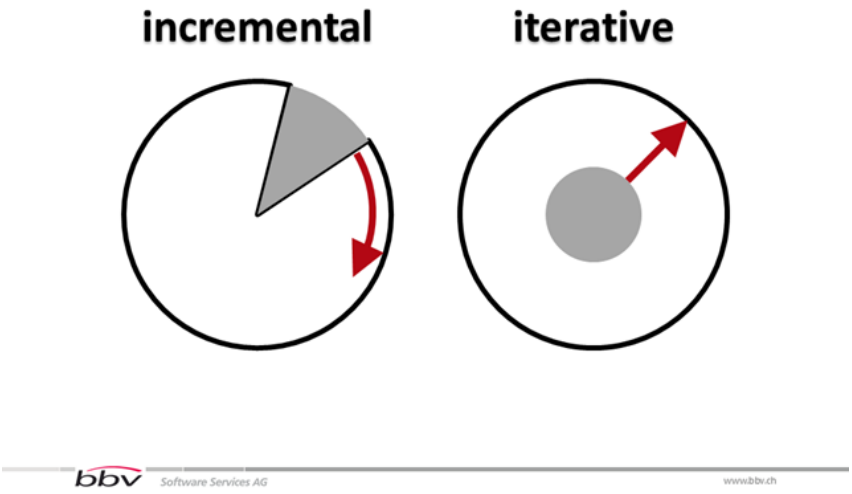
Yhden tai useamman ketterän menetelmän soveltaminen Scrumin kanssa on yleistä, sillä se ei yksinään välttämättä kata koko projektin elinkaarta. Tästä johtuen yritykset usein lisäävät ylimääräisiä vaiheita luodakseen kattavamman implementaation. Esimerkiksi projektin alkuun on tavanomaista lisätä vaiheita, kuten ohjeistusta vaatimuksien keräämiseen ja priorisointiin, alustavaa korkean-tason suunnittelua, sekä budjetin ja aikataulun ennustuksia. [6.]

2.3 Toimintaperiaatteet

Scrum on viitekehys, mikä on luotu helpottamaan ihmisiä ratkaisemaan monimutkaisia ongelmia kehittäessään tuotteita tuottavasti ja luovasti mahdollisimman korkealla lisäarvolla. Scrum on kevyt ja helppo ymmärtää, mutta sitä on vaikea hallita hyvin. Sitä on hyödynnetty tuotteiden kehityksessä jo 1990-luvun alusta lähtien. Kyse ei ole tuotekehitysprosessista tai -tekniikasta, vaan viitekehyksestä, minkä sisällä voidaan käyttää useita erilaisia prosesseja sekä tekniikoita. Scrumilla tuotehallinnon ja kehityksen menetelmien vaikutukset tuodaan näkyville, jotta menetelmiä voidaan parantaa.

Scrum koostuu Scrum-tiimeistä rooleineen, tapahtumista, tuotoksista ja säännöistä. Jokainen osa palvelee tiettyä tarkoitusta ja on oleellinen osa Scrumin onnistumista. Eri roolit, tapahtumat ja tuotokset sidotaan yhteen Scrum-säännöillä, joilla ohjataan näiden välisiä vuorovaikutuksia. Säännöt esitellään tarkemmin myöhemmin raportissa.

Scrum pohjautuu empiiriseen prosessinhallintateoriaan, tai empirismiin, minkä mukaan tieto perustuu kokemukseen ja päätökset tehdään tunnettujen tosiasioiden pohjalta. Scrumissa hyödynnetään iteratiivis-inkrementaalista, eli toistavaa ja lisäävää, lähestymistapaa ennustettavuuden optimointiin ja riskien kontrollointiin (kuva 1). Empiirinen prosessinhallinta pohjautuu kolmeen pääajatukseen, jotka ovat läpinäkyvyys, tarkastelu ja sopeuttaminen. [2.]



Kuva 1. Scrumissa työskennellään toistavasti ja lisäävästi [18.]

2.3.1 Läpinäkyvyys

Prosessiin vaikuttavien tekijöiden on oltava helposti havaittavissa niille, jotka ovat vastuussa lopputuloksesta. Läpinäkyvyys edellyttää, että prosessin kannalta merkittävät tekijät määritellään sekä sovitaan yhdessä, jotta saadaan tarkastelijoille yhteinen näkemys tarkasteltavasta asiasta. Tämä edellyttää esimerkiksi sitä, että kaikilla osallisilla on yhteinen prosessiin viittaava sanasto sekä että työn tekijöillä kuin hyväksyjillä on yhteinen määritelmä ”valmiista”. [2.]

2.3.2 Tarkastelu

Haitallisten poikkeamien havaitsemiseen ajoissa Scrumin käyttäjien on säännöllisesti tarkastettava Scrum-tuotoksia ja työn edistymistä kohti tavoitetta. Tätä ei saa kuitenkaan tapahtua niin usein, että se vaikuttaisi häiritsevästi varsinaiseen työntekoon. Suurin hyöty tarkastelulla saavutetaan, kun tarkastuksen hoitaa ammattitaitoinen tarkastelija sopivin väliajoin. [2.]

2.3.3 Sopeuttaminen

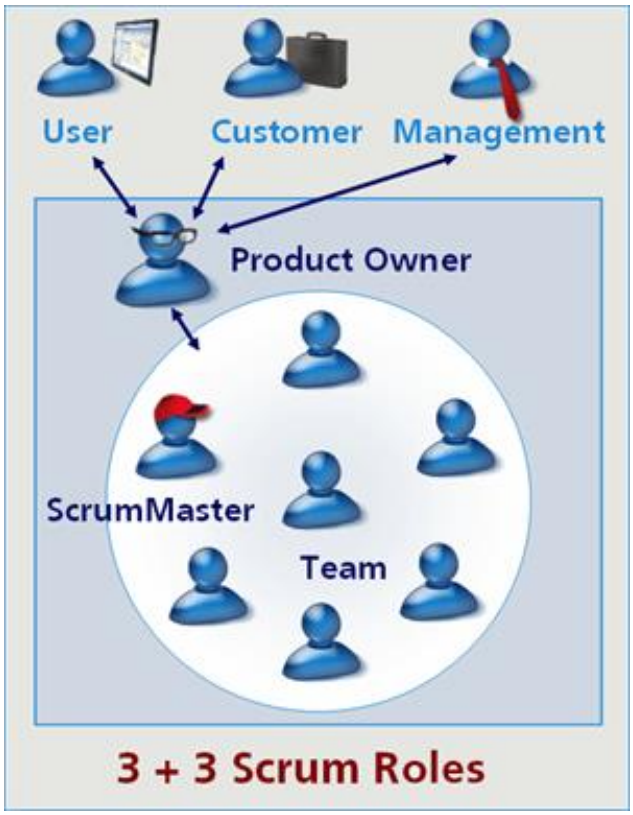
Tarkastelijan huomattua, että yksi tai useampi prosessin osa on hyväksyttävien raja-arvojen ulkopuolella, eikä täten syntyvää tuotetta voisi hyväksyä, tarkastelijan tulee säätää prosessia tai käytössä olevia materiaaleja. Tämä tulee tehdä niin nopeasti kuin mahdollista, jotta voidaan minimoida myöhemmät poikkeamat. [2.]

Scrum pitää sisällään neljä muodollista kohtaa tarkasteluun ja sopeuttamiseen, joita ovat sprintin suunnittelupalaveri, päiväpalaveri, sprinttikatselmus sekä sprintin retrospektiivi. Näitä aiheita käsitellään myöhemmin raportissa. [2.]

2.4 Scrum-tiimi

Scrum-tiimi muodostuu scrum-masterista, kehitystiimistä ja tuotteenomistajasta (kuva 2). Kehitystiimi on itseohjautuva ja monitaitoinen. Itseohjautuvan tiimin etu on siinä, että he voivat itse päättää parhaan tavan tehdä työnsä ulkoisen ohjauksen sijaan. Tämän lisäksi monitaitoisilla tiimeillä on kaikki työhön vaadittava osaaminen ilman riippuvuuksia tiimin ulkopuolisiin henkilöihin. Scrumin tiimimalli on suunniteltu joustavuuden, luovuuden ja tuottavuuden optimoimiseksi.

Tuotteita kehitetään ja toimitetaan scrum-tiimin toimesta toistavasti ja lisäävästi. Täten maksimoidaan tilaisuudet palautteen saamiseen. Vähittäin kasvavan ”valmiin” tuotteen toimitukset pitävät huolen siitä, että tuotteesta saadaan aina toimiva ja mahdollisesti hyödyllinen versio. [2.]



Kuva 2. Scrumin roolit [19.]

2.4.1 Tuoteomistaja

Tuoteomistajan vastuulla on tuotteen ja kehitystiimin työn arvon maksimoiminen. Tuotteen omistaja on se, joka viime kädessä vastaa tuotteen ominaisuuksista (kuva 3). Tuoteomistajan vastuualueisiin kuuluvat myös tuotteen kehitysjonon hallinta ja sen eri kohtien selkeä kirjallinen ilmaiseminen. Hänen kuuluu järjestää kehitysjonon kohdat siten, että tavoitteisiin päästään parhaalla mahdollisella tavalla. Hänen tulee ylläpitää kehitysjonon avoimuutta, läpinäkyvyyttä ja ymmärrettävyyttä siten, että kehitysjonosta selviää, mitä scrum-tiimin tulee seuraavaksi tehdä. Tuoteomistajan tulee myös pitää huoli siitä, että kehitystiimi ymmärtää kehitysjonon kohdat riittävällä tarkkuudella.

Tuoteomistaja voi tehdä itse kaiken edellä mainitun tai pyytää kehitystiimiä tekemään ne. Tuoteomistaja kantaa kuitenkin vastuun siitä, että kaikki tulee tehdyksi.

Kuva 3. Tuoteomistaja omistaa projektin vision ja edustaa asiakasta [20.]

Tuoteomistaja on yksi henkilö. Hän voi hyödyntää tai edustaa komiteoiden toiveita kehitysjonon kautta, mutta kehitysjonon muuttamiseksi on aina ensin vakuutettava tuoteomistaja.

Koko organisaation tulee kunnioittaa tuoteomistajan päätöksiä. Päätökset ovat nähtävissä kehitysjonon sisällössä ja prioriteeteissa. Kenelläkään muulla kuin tuoteomistajalla ei ole valtaa käskeä kehitystiimiä työskentelemään muiden kuin tuoteomistajan tämän asettamien vaatimusten parissa. Kehitystiimin tulee myös jättää huomioimatta tällaiset pyynnöt. Tuoteomistaja ei normaalisti saa olla yksi kehitystiimin jäsenistä. Aloittelevassa Scrum kehityksessä tuoteomistajana tulee olla vain yksi henkilö, joskin tälle tulee olla myös sijainen tarvittaessa. [7.]

2.4.2 Kehitystiimi

Kehitystiimi on ryhmä ammattilaisia, jotka muuttavat kehitysjonon sisällön mahdollisesti julkaisukelpoiseksi ”valmiiksi” tuoteversioksi jokaisessa sprintissä. Tuoteversion kehitykseen osallistuvat ainoastaan kehitystiimin jäsenet.

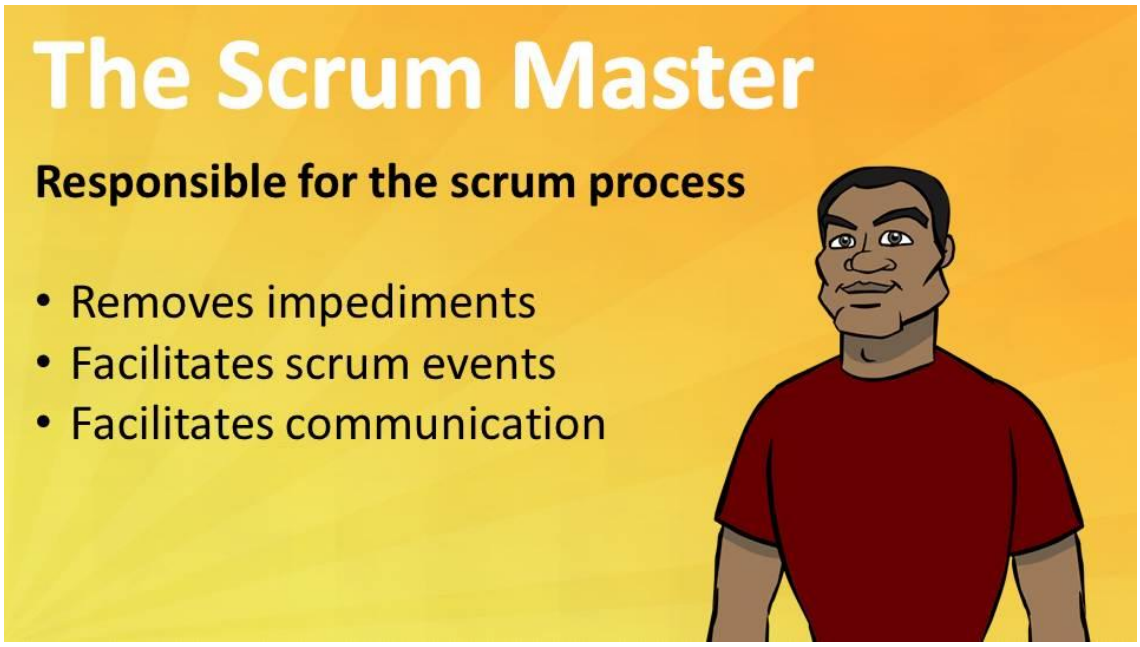
Yritykset muodostavat ja valtuuttavat kehitystiimit hallitsemaan ja hoitamaan omaa työtään, minkä on todettu parantavan kehitystiimin suorituskykyä ja tuottavuutta.

Kehitystiimeille olennaista on niiden itseohjautuvuus. Heillä on vapaat kädet sen suhteen, kuinka tuotteen kehitysjono muutetaan julkaisukelpoiseksi tuoteversioksi. Tiimi muodostuu monitaitoisista jäsenistä, joilla on kaikki tarvittava osaaminen julkaisukelpoisen tuoteversion kehittämiseen. Jokainen jäsen kantaa poikkeuksetta titteliä ”kehittäjä” huolimatta siitä, millaista työtä kyseinen henkilö tekee. Vaikka tiimin jäsenillä voi olla erityistä osaamista ja työn painopisteet voivat vaihdella, on vastuu kehityksen kulusta koko tiimillä yhteinen. Kehitystiimit eivät voi sisältää alitiimejä, jotka voisivat vastata eri osa-alueista, kuten testaamisesta tai liiketoiminta-analyysistä.

Kehitystiimin koko on pyrittävä pitämään riittävän pienenä ja ketteränä, mutta kuitenkin tarpeeksi suurena, että työtä saadaan valmiiksi merkittäviä määriä. Liian pieni tiimi, alle kolme jäsentä, saattaa heikentää vuorovaikutteisuutta sekä tuottavuushyötyä. Tämän lisäksi sprintin aikana on mahdollista törmätä osaamispulaan jollakin osa-alueella, jolloin julkaisukelpoisen tuoteversion tuottaminen vaarantuu. Kehitystiimin kasvaessa liian suureksi, enemmän kuin yhdeksän jäsentä, resursseja kuluu liikaa työn koordinointiin. Tuoteomistajaa ja scrum-masteria ei lasketa näihin lukuihin mukaan, elleivät he ole osa kehitystiimiä. Pyrkimys on että jokaisen tiimin jäsenen allokointi tiimin työhön olisi 100%. Lisäksi tiimin henkilöiden tulisi pysyä samoina koko projektin ajan. [2.]

2.4.3 Scrum-master

Scrum-masterin tehtävä on pitää huoli siitä, että kaikki ymmärtävät ja käyttävät Scrumia. Tämän hän saavuttaa varmistamalla, että scrum-tiimit pitäytyvät Scrumin teoriassa, käytännöissä ja säännöissä. Scrum-master on scrum-tiimin palveleva johtaja. Hänen on autettava scrum-tiimin ulkopuolisia ymmärtämään, miten heidän kannattaa toimia scrum-tiimin kanssa, jotta voidaan maksimoida scrum-tiimin työn arvo (kuva 4). [2.]



Kuva 4. Scrum-masterin vastuualueita [21.]

Scrum-master voi neuvoa tuoteomistajaa kehitysjonon hallinnassa. Hänen tulee välittää kehitystiimille selkeä kuva tuotteen visiosta, tavoitteista ja kehitysjonon kohdat. Hänen tulee opettaa kehitystiimiä luomaan selkeitä ja ytimekkäitä kehitysjonon kohtia. Scrum-masterin tulee myös poistaa mahdolliset esteet kehitystiimin etenemisen tieltä, ja on opastettava näitä itseohjautuvuudessa. Scrum-master tarkkaillee työn etenemistä. Jos hän huomaa, että tavoitteita ei voida saavuttaa, on hänen kommunikoitava asiasta tuoteomistajan ja kehitystiimin kanssa tilanteen korjaamiseksi tai sprintin sisällön kaventamiseksi. Scrum-master pitää huolen siitä, että tiimiä ei häiritä sprintin aikana uusilla vaatimuksilla ja pyrkii turvaamaan kehitystiimin työrauhan. [6.]

2.5 Scrumin tapahtumat

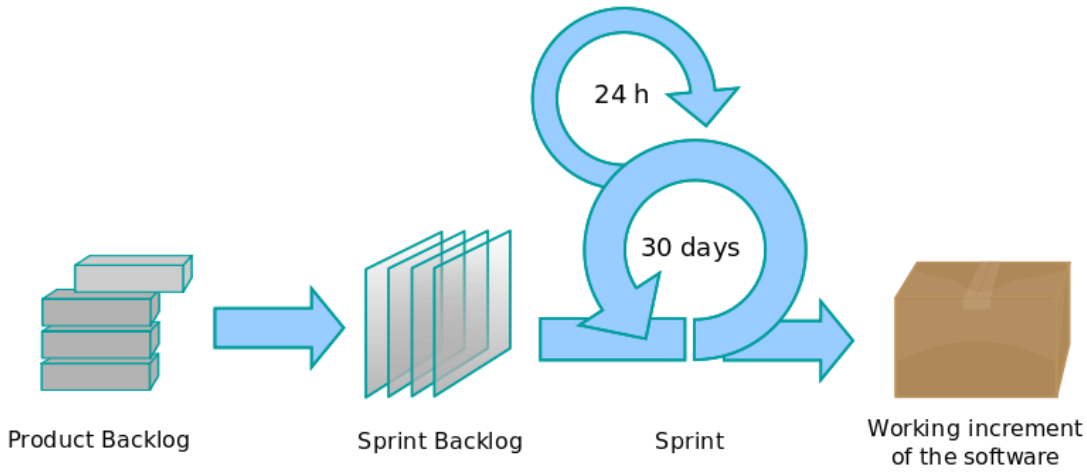
Scrumissa oleellista ovat ennalta sovitut tapahtumat, joilla luodaan säännöllisyyttä ja minimoidaan muiden kuin Scrum-palavereiden tarve. Jokainen tapahtuma on aikarajattu, millä varmistetaan, että suunnittelulle varataan riittävästi aikaa, mutta ei pääse syntymään hukkaa. Jokainen tapahtuma mahdollistaa tuotteen tarkastelun ja sopeuttamisen Sprintin ollessa ainoa poikkeus, mikä sisältää muut tapahtumat. Tapahtumat on suunniteltu siten, että ne lisäävät tuotekehitykselle olennaista läpinäkyvyyttä sekä mahdollistamaan sen tarkastelua. Tapahtumien pois jättäminen vähentää läpinäkyvyyttä ja johtaa menetettyihin tilaisuuksiin tarkastelussa ja sopeuttamisessa. [2.]

2.5.1 Sprintti

Sprint on Scrumin sydän, joka on enintään kuukauden mittainen tai sitä lyhyempi aika. Sen aikana tuotetaan ”valmiin” määritelmän täyttävä, käyttö- sekä julkaisukelpoinen tuoteversio. Sprintin pituus pysyy vakiona koko kehitysprosessin ajan. Uusi sprintti aloitetaan heti edellisen päätyttyä.

Sprintti koostuu sprintin suunnittelupalaverista, päiväpalaverista, kehitystyöstä, sprinttikatselmuksesta ja sprintin retrospektiivistä. Sprintin aikana ei ole sallittua tehdä muutoksia, jotka voisivat vaikuttaa sprintin tavoitteeseen ja kehitystiimin koostumus on pidettävä samana. Laatutavoitteissa on pysyttävä, mutta sprintin sisältö on tarkennettavissa ja neuvoteltavissa tuoteomistajan ja kehitystiimin kanssa, kun ratkaistavasta ongelmasta on opittu enemmän.

Jokaista sprinttiä voidaan ajatella pienenä projektina. Yhteistä molemmilla on se, että niitä käytetään jonkin tietyn tavoitteen saavuttamiseen. Jokainen sprintti käsittää määritelmän, mitä ja miten tullaan toteuttamaan, joustavan suunnitelman, joka ohjastaa toteutusta sekä varsinaisen työn ja sen tuloksena kehittyvän tuoteversion (kuva 5).



Kuva 5. Scrum-prosessi [22.]

Sprinttien ollessa enintään yhden kalenterikuukauden mittaisia vältytään siltä, että toteutettavan sisällön määritelmä muuttuisi kesken kaiken. Tällöin myös työn monimutkaisuus ja riskit saadaan pysymään matalampina. Sprinttien käyttö helpottaa työn ennustettavuutta mahdollistamalla työn edistymisen seuraamisen ja sopeuttamisen kohti tavoitetta ainakin kerran kuukaudessa. Riskit rajoittuvat enimmillään kuukauden kustannuksiin, sprintin maksimipituuden myötä.

Sprintti on keskeytettävissä ennen aikarajansa päättymistä, mutta vain tuoteomistajalla on valta tähän. Sidosryhmillä, kehitystiimillä ja scrum-masterilla on kuitenkin mahdollisuus vaikuttaa tuoteomistajan päätökseen omilla näkemyksillään asiasta.

Sprint on keskeytettävissä, jos huomataan sen tavoitteen muuttuneen tarpeettomaksi. Tämä voi olla seurausta yrityksen suunnan muutoksesta, jos markkinat tai teknologiset edellytykset muuttuvat. Keskeyttäminen on silloin suotavaa, kun voidaan todeta, että sprintin loppuun vienti ei ole enää kannattavaa. Keskeyttäminen kannattaa kuitenkin hyvin harvoin sprinttien lyhyestä kestosta johtuen.

Keskeytyksen tapahduttua tulee mahdolliset ”valmiin” määritelmän täyttävät kehitysjonon kohdat katselmoida. Jos osasta löytyy potentiaalia julkaisuun asti, tuoteomistaja usein hyväksyy ne. Myös keskeneräisiksi jääneet kehitysjonon kohdat uudelleen arvioidaan ja siirretään takaisin kehitysjonoon. Koska keskeneräinen työ vanhenee nopeasti, on ne uudelleen arvioitava säännöllisin väliajoin.

Keskeyttäminen kuluttaa resursseja, sillä tällöin kaikkien on osallistuttava uuteen sprintin suunnittelupalaveriin, jotta uusi sprintti voitaisiin aloittaa. [2.]

2.5.2 Sprintin suunnittelupalaveri

Suunnittelupalaverissa päätetään sprintin aikana toteutettavat työt. Suunnitelma luodaan yhdessä koko scrum-tiimin voimin. Suunnittelupalaveri on rajattava pituudeltaan enintään kahdeksaan tuntiin käytettäessä kuukauden mittaisia sprinttejä. Tätä lyhyemmille sprinteille varataan suhteessa vähemmän aikaa.

Suunnittelupalaveri koostuu kahdesta elementistä, jotka ovat pituudeltaan enintään puolet koko suunnittelupalaverin pituudesta. Näiden osien aikana on löydettävä vastaus siihen, mitä toimitetaan alkavan sprintin tuoteversiossa ja miten se voidaan toteuttaa.

Palaverin ensimmäisessä osassa kehitystiimi luo ennusteen toiminnallisuuksista, jotka voidaan toteuttaa sprintin aikana. Tuoteomistaja esittelee järjestetyn kehitysjonon, jonka pohjilta scrum-tiimi pohtii yhdessä, mitkä toiminnallisuudet kehitysjonosta valitaan mukaan sprinttiin.

Lähtökohtana palaverille on tuotteen järjestetty kehitysjono, viimeisin tuoteversio, jos sellainen on, kehitystiimin kapasiteetti ja aikaisempi suorituskyky. Kehitystiimi tekee yksin päätöksen sopivasta työmäärästä, koska vain he itse voivat arvioida, mitä he pystyvät alkavassa sprintissä toteuttamaan. Kun kehitysjonon kohdat on valittu, valitaan sprintille tavoite. Tämä muodostuu sprinttiin valittujen kehitysjonon kohtien toteutuksen kautta. Tavoitteen asettaminen antaa kehitystiimille selkeän näkökulman ja perustelun sille, miksi se on kehittämässä tuoteversiota.

Jälkimmäisessä vaiheessa kehitystiimi suunnittelee valittujen kohtien pohjalta, miten työ toteutetaan ”valmiiksi” tuoteversioksi. Kehitysjonon kohdista ja suunnitelmista niiden toteuttamiseksi muodostuu sprintin tehtävälista.

Ensimmäiseksi kehitystiimi suunnittelee toiminnallisuudet ja työt, jotka ovat oleellisimmat toimivan tuoteversion kehittämisessä. Vaikka töiden koot voivat vaihdella, pyrkii kehitystiimi valitsemaan sopivan määrän työtä, joka on toteutettavissa sprintin loppuun mennessä. Palaverin loppuun mennessä pyritään pilkkomaan ensimmäisille päiville suunnatut työt enintään päivän mittaisiin yksiköihin. Kehitystiimin tulee itseohjautua niin suunnittelupalaverissa kuin sprintin aikana saavuttaakseen tehtävälistan tavoitteet.

Tuoteomistaja saattaa olla läsnä myös suunnittelupalaverin toisella puoliskolla. Tällöin hän voi tarkentaa kehitystiimille kehitysjonon kohtia tai auttaa näitä löytämään kompromisseja. Kehitystiimin huomatessa, että sprintti on joko liian työläs tai kevyt, voivat he neuvotella uudestaan asiasta tuoteomistajan kanssa. Kehitystiimillä on myös oikeus kutsua muita paikalle saadakseen joko teknistä tai liiketoiminnallisia neuvoja.

Palaverin loppuun mennessä kehitystiimin on kyettävä selittämään tuoteomistajalle ja scrum-masterille, kuinka he aikovat saavuttaa sprintin tavoitteen ja luoda ”valmiin” tuoteversion.

Tavoitteen asettaminen sprintille antaa kehitystiimille hieman liikkumavaraa toteutuksen suhteen. Toiminnallisten ja teknologisten ongelmien ratkaiseminen on kehitystiimin ainoa keino saavuttaa sprintin tavoite. Jos tässä ei onnistuta, tulee kehitystiimin olla yhteydessä tuoteomistajaan ja toteuttaa sprintin toiminnallisuudet vain osittain. [2.]

2.5.3 Päiväpalaveri

Päiväpalaveri on 15 minuuttiin rajattu tapahtuma, jossa tahditetaan kehitystiimin keskinäiset työt ja tehdään suunnitelma seuraaville 24 tunnille. Tapahtumassa käydään läpi edellisen päiväpalaverin jälkeen tehdyt työt ja pyritään ennustamaan, mitä on toteutettavissa ennen seuraavaa päiväpalaveria. Päiväpalaverit pidetään aina samaan aikaan ja samassa paikassa.

Päiväpalaverissa jokainen kehitystiimin jäsen kertoo lyhyesti, mitä on saanut aikaiseksi viime tapaamisen jälkeen ja mitä aikoo tehdä seuraavaksi. Mahdolliset työn etenemistä haittaavat esteet on myös tuotava esille.

Päiväpalaverin tarkoitus on myös pitää kehitystiimi tietoisena työn edistymisestä kohti sprintin tavoitteita sekä tehtävälistan toteutumisesta. Kehitystiimillä on mahdollisuus tavata päiväpalaverin jälkeen ja suunnitella tarkemmin sprintin jäljellä olevaa työtä. Tuoteomistajalle ja scrum-masterille olisi kyettävä selvittämään päivittäin, kuinka he aikovat saavuttaa sprintin tavoitteen ja odotetun tuoteversion jäljellä olevassa ajassa.

Kehitystiimin vastuulla on päiväpalaverin järjestäminen ja scrum-masterin tulee varmistaa, että se myös pidetään. Scrum-masterin tulee ohjeistaa kehitystiimiä pitämään palaveri lyhyenä ja valvoa sääntöä, että palaveriin ei osallistu ulkopuolisia. Päiväpalaveri on suunnattu ainoastaan kehitystiimille, kyseessä ei siis ole raportointipalaveri.

Päiväpalavereilla parannetaan tiimin kommunikointia, vähennetään tarvetta muille palavereille, autetaan tunnistamaan ja poistamaan kehitystä haittaavia esteitä sekä nopeutetaan päätöstentekoa ja nostetaan kehitystiimin asiantuntemusta. [2.]

2.5.4 Sprinttikatselmus

Sprintin päätteeksi pidetään sprinttikatselmus, missä käydään läpi kehitetty tuoteversio ja tarvittaessa sopeutetaan tuotteen kehitysjonoa. Katselmoinnissa scrum-tiimi ja sidosryhmät käyvät läpi yhdessä, mitä sprintissä kehitettiin. Tämän pohjalta ja mahdollisten sprintin aikana tapahtuneiden kehitysjonoon kohdistuneiden muutosten pohjalta osallistujat pohtivat, mitä kehitetään seuraavaksi. Tuoteversiota demotaan ja tavoitteena on saada palautetta, mistä saadaan pohja seuraavalle sprintin suunnittelupalaverille. Kuukauden mittaiselle sprintille varataan katselmukseen maksimissaan 4 tuntia aikaa. Lyhyemmille sprinteille tulee varata suhteessa vähemmän aikaa.

Sprinttikatselmuksissa tuoteomistaja tunnistaa mikä on ja ei ole ”valmista”. Kehitystiimi käy läpi, mikä sprintissä meni hyvin, millaisia ongelmia kohdattiin ja kuinka ne ratkaistiin. Kehitystiimi esittelee katselmuksessa ”valmiin” työn ja vastaa tuoteversioon liittyviin kysymyksiin. Tämän jälkeen tuoteomistaja kertoo kehitysjonon tilanteen ja antaa arvion valmistumisajankohdasta tähänastiseen edistymiseen pohjaten. Lopuksi käydään läpi, mitä on kannattavaa tehdä seuraavaksi, jotta saadaan mahdollisimman hyvä pohja seuraavaan sprintin suunnittelupalaveriin. Sprinttikatselmuksen myötä tuotteen kehitysjono on tarkistettu ja sisältää todennäköiset kohdat seuraavaa sprinttiä varten. [2.]

2.5.5 Sprintin retrospektiivi

Sprinttikatselmuksen jälkeen pidetään sprintin retrospektiivi. Sprintin retrospektiivissä scrum-tiimi saa mahdollisuuden tarkastella omaa työskentelyään ja tehdä suunnitelmia kehitysprosessin parantamiseen. Palaveri on rajattu maksimissaan kolmeen tuntiin kuukauden mittaiselle sprintille. Sitä lyhyemmille sprinteille varataan suhteessa vähemmän aikaa.

Retrospektiivissä tarkastellaan, miten sujuvasti sprintti meni ja pyritään määrittelemään parannuksia kohtiin, joissa oli ongelmia, sekä luodaan suunnitelma näiden toteuttamiseksi.

Sprinttien tuottavuuden ja miellyttävyyden nostamiseksi scrum-masterin tulee kannustaa kehitystiimiä tarkastelemaan ja sopeuttamaan Scrum-viitekehykseen liittyvää kehitysprosessia ja käytäntöjä. Retrospektiivissä kehitystiimin käy läpi tapoja laadun nostattamiseksi ja sopeuttaa tarvittaessa ”valmiin” määritelmää.

Retrospektiivi on siis muodollinen tilaisuus kehitysprosessin tarkastelua ja sopeuttamista varten. Se auttaa kehitystiimiä tunnistamaan parannusmahdollisuuksia, mitkä sen tulee käyttää seuraavassa sprintissä. Seuraavaan sprinttiin valittujen parannusten toteuttaminen sopeuttaa tiimiä sen itseensä suuntaamaan tarkasteluun. [2.]

2.6 Scrumin tuotokset

Scrumin tuotoksilla pyritään kuvaamaan työtä tai lisäarvoa keinoilla, joilla voidaan lisätä läpinäkyvyyttä ja tilaisuuksia tarkastelulle sekä sopeuttamiselle. Ne onkin suunniteltu nimenomaan maksimoimaan läpinäkyvyys niille tiedoille, jotka helpottavat kehitystiimiä tuottamaan ja toimittamaan ”valmiin” tuoteversion. [2.]

2.6.1 Tuotteen kehitysjono

Tuotteen kehitysjonolla tarkoitetaan järjestettyä listaa, mikä sisältää kaiken, mitä tuotteessa saatetaan tarvita. Se toimii ainoana lähteenä toteutettaville vaatimuksille ja muutoksille. Tuoteomistaja huolehtii kehitysjonon sisällöstä, saatavuudesta ja järjestämisestä. Kehitysjono ei valmistu koskaan. Kehitysjono elää koko projektin ajan, ensimmäinen versio sisältääkin ainoastaan alustavan listan parhaiten tunnetuista vaatimuksista. Kehitysjono siis kehittyy koko ajan ja muovautuu muistuttamaan enemmän sitä, mitä tuotteelta vaaditaan ollakseen tarkoituksenmukainen, kilpailukykyinen ja käyttökelpoinen. Kehitysjono on olemassa koko tuotteen elinkaaren ajan.

Kehitysjonoon on listattu kaikki ominaisuudet, toiminnot, vaatimukset, parannukset ja korjaukset, jotka on tarkoitus toteuttaa tulevissa tuoteversioissa. Nämä järjestellään yleensä arvon, riskin, prioriteetin ja välttämättömyyden mukaan. Mitä korkeammalla kohta on, sitä välittömämpi sen kehitystarve on ja sitä enemmän sitä on ehditty suunnitella.

Sijainti kehitysjonossa kuvaa suoraan sitä, miten selkeästi ja yksityiskohtaisesti kohtaa on kuvattu. Ylempänä oleviin kohtiin on helpompi arvioida työmäärät kuin alempana oleviin näiden selkeyden ja yksityiskohtien määrän takia. Sprinttiin valittavat kohdat on hajoitettava niin pieniin osiin, että jokainen valittu kohta on tehtävissä ”valmiiksi” sprintin loppuun mennessä. Kehitysjonon kohdat, jotka ovat toteutettavissa yhden sprintin aikana, kutsutaan ”sprinttiin sopiviksi” sprintin suunnittelupalaverissa.

Kehitysjono muuttuu kattavammaksi listaksi sitä mukaan, kun tuote otetaan käyttöön ja sen arvo kasvaa ja markkinoilta saadaan palautetta. Vaatimukset elävät koko ajan, johon vaikuttavat mm. muutokset liiketoiminnassa, markkinoissa ja teknologiassa.

Kehitysjonoa työstetään toistuvasti yhdessä tuoteomistajan ja scrum-tiimin kanssa. Tällöin he lisäävät kehitysjonon kohtiin lisää yksityiskohtia ja järjestelevät kehitysjonoa uudelleen. Työstön aikana kehitysjonon kohdat katselmoidaan ja arvioidaan uudelleen. Näiden päivittäminen on kuitenkin mahdollista milloin tahansa tuoteomistajan toimesta tai toiveesta. Työstöstä seuranneiden muutosten käytännön toteutuksesta päättää kehitystiimi itse. Kehitystiimi tekee kaikki työmääräarviot. Tuoteomistaja voi kuitenkin auttaa heitä ymmärtämään vaatimuksia ja tekemään kompromisseja.

Jäljellä oleva kokonaistyömäärä lasketaan tuoteomistajan toimesta vähintään jokaisessa sprintkatselmuksessa ja vertaa saamiaan lukuja edellisten sprintkatselmusten lukuihin. Vertailulla pystytään arvioimaan työn edistymisnopeutta kohti tavoitetta ja toivotun aikarajan toteutumista. Nämä arviot ovat kaikkien nähtävillä. [2.]

2.6.2 Sprintin tehtävälista

Sprintin tehtävälista koostuu kehitysjonosta valituista kohdista ja suunnitelmista niiden toteuttamiseksi. Tehtävälista on kehitystiimin luoma arvio seuraavan tuoteversion sisältämistä toiminnallisuuksista sekä siihen vaadittavasta työmäärästä. Tehtävälistalla määritetään siis työt, jotka kehitystiimin tulee tehdä luodakseen kehitysjonon kohdista ”valmiin” tuoteversion.

Tehtävälistan tulee olla riittävän yksityiskohtainen, jotta päiväpalaverissa voidaan havaita muutokset työn edistymisestä. Kehitystiimi muokkaa tehtävälistaa tarvittaessa koko sprintin ajan pitäen mielessä mikä on oleellista sprintin tavoitteiden saavuttamiseksi. Tehtävälistan lopullinen muoto selviääkin tästä syystä vasta sprintin lopussa.

Jos tehtävälistasta puuttuu jotain tavoitteen saavuttamiseksi, se lisätään sinne, tai jos jokin tehtävä todetaan turhaksi, se voidaan poistaa. Ainoastaan kehitystiimillä on oikeus muuttaa tehtävälistan sisältö sprintin aikana. Arvioitua jäljellä olevaa työaikaa päivitetään töiden edetessä ja tehtävien valmistuessa. Tehtävälistä on päivittyvä kuva siitä työstä, jonka kehitystiimi pyrkii saamaan valmiiksi sprintin aikana (kuva 6).

Omaa edistymistään sprintissä kehitystiimi saa seurattua vähintään jokaisessa päiväpalaverissa päivittämästään kokonaistyönmäärästä. Vertaamalla tätä lukua aiempiin päiviin voidaan arvioida todennäköisyys sprintin tavoitteen saavuttamiseksi. Ainoat merkittävät muuttujat tätä selvittäessä ovat jäljellä oleva työmäärä sekä päivämäärä.

Tuoteversio koostuu kehitysjonon kohdista, jotka ovat valmistuneet sprinttien aikana. Sprintin lopussa olevan version tulee olla ”valmis” ja oltava käyttökelpoisessa kunnossa siitä huolimatta, aikooko tuoteomistaja julkaista sen. [2.]

Kuva 6. Esimerkki Sprintin tehtävälistasta [23.]

2.7 ”Valmiin” määritteleminen

Kun kehitysjonon kohtaa tai tuoteversiota kutsutaan ”valmiiksi”, jokaisen on ymmärrettävä, mitä sillä tarkoitetaan (kuva 7). Määritelmät vaihtelevat eri scrum-tiimien välillä, mutta jokaisella tiimiin kuuluvalla tulee kuitenkin olla yhteinen käsitys termistä, jotta läpinäkyvyys on turvattu. Määritelmä on siis yhdessä tuoteomistajan ja kehitystiimin laatima laadullinen kriteeristö toteutettaville toiminnallisuuksille. Määritelmän avulla arvioidaan, milloin tuoteversioon liittyvä työ on valmista.

Määritelmän avulla kehitystiimin on helpompi arvioida, montako kohtaa kehitysjonosta voidaan valita sprintin suunnittelupalaverissa. Jokaisessa sprintissä on tavoitteena julkaista toimiva tuoteversio, joka noudattaa uusinta tuotettua ”valmiin” määritelmää. Jokaisen tuoteversion ollessa käyttökelpoinen, on tuoteomistajasta kiinni, julkaistaanko se välittömästi vai ei. Tuoteversiot rakentuvat aikaisempien versioiden päälle ja ovat läpikotaisin testattuja. Näin ollen voidaan olla varmoja tuoteversioiden yhteensopivuudesta.

Scrum-tiimien kehittyessä ”valmiin” määritelmä kehittyy heidän mukana taaten tiukemmat kriteerit korkealle laadulle. [2.]

Kuva 7. Valmiin määritelmä [24.]

3 Knights Quest

3.1 Pelin idea ja sen synty

Vuonna 2008 vanhimman veljeni ollessa vielä töissä NSD:llä (National Software Development, nyk. Cybercom) järjestivät he koulutuksen, missä hänelle ja hänen kollegoilleen opetettiin Scrumin peruskäytäntöjä. Harjoitukseksi he saivat ryhmätehtävän, missä heidän tuli tuottaa peli. Ryhmille ei annettu valmiita ideoita, vaan ryhmän tehtävänä oli kehittää koko peli alusta asti. Aihetta tai alustaa ei rajattu mitenkään, vaan peli saattoi olla korttipeli, tietokonepeli tai vaikka lautapeli.

Näihin aikoihin pelasimme paljon Settlers of Catan nimistä lautapeliä. Pelin kartta koostuu 37-stä kuusikulmiosta, eli heksasta (kuva 8), jotka asetetaan pelin alussa pöydälle satunnaisessa järjestyksestä. Veljeni ehdottikin tätä pelilaudan kasaamistapaa yhdistettynä vanhaan pc- ja amiga-klassikkoon Moonstoneen. Moonstone: A Hard Days Knightista ajatuksena oli ottaa mukaan pelin vuoropohjainen kartalla liikkuminen, varusteiden ostaminen ja taistelu (kuva 8). Idea hyväksyttiin, koska pelit olivat ennestään tuttuja ryhmän muillekin jäsenille ja se vaikutti suhteellisen helpolta toteuttaa annetussa aikamääreessä. Koska pelistä puuttui vielä varsinainen päämäärä, keksi veljeni ottaa vielä lautapelistä Menolippu mukaan pisteytys- ja tavoite järjestelmän, jolla ratkaistaisiin pelin voittaja ja päättyminen. Pelin alussa pelaajille annettaisiin muilta pelaajilta salassa pidettävät päämäärät ja pisteitä kertyisi näitä päämääriä saavuttamalla. Tässä vaiheessa veljelläni oli jo melko selkeä mielikuva siitä, mihin pyritään ja millainen kokonaisuus siitä muodostuisi. Hänen tehtäväkseen muodostui määrittelijän tai arkkitehdin rooli, muiden vastatessa toteutuksesta. Muillakin oli tietenkin oikeus ilmaista mielipiteensä, jos huomasi epäkohtia suunnitelmassa. Ideasta muodostui lopulta lautapeliksi sovellettu prototyyppi. [8; 9.]

Kuva 8. Vasemmalla taisteluruutu pelistä Moonstone, oikealla Settlers of Catan lautapelikartta [8. 9.]

Koska idea tuntui veljestäni hyvältä vielä useamman vuoden jälkeenkin, niin hän ajatteli, että sen voisi toteuttaa oikeastikin. Vuoden 2012 kevättalvella hän esitteli ideansa ja kysyi, jos minua kiinnostaisi toteuttaa se hänen kanssaan ja että voisin mahdollisesti saada sen hyväksytyksi lopputyönä koululle. Koska idea kuulosti hyvältä ja pelien tekeminen työksi kiinnosti, ajattelin, että saisin siitä samalla hyvän esimerkkityön portfoliooni. Niinpä suostuin mukaan pelin kehittämiseen.

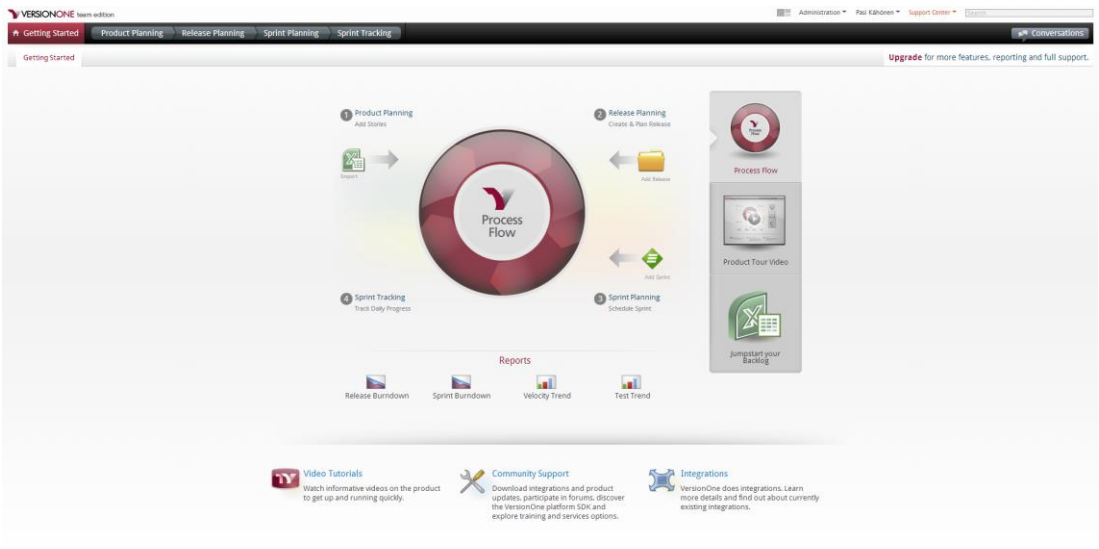
Alkuperäisidea on pysynyt hyvin pitkälti koskemattomana videopelimuotoon siirrettäessä, joitain muutoksia on kehityksen myötä kuitenkin tehty ja tullaan vielä tekemään ennen kuin peli on julkaistu. Pelin lyhyt määritelmä on seuraava: kartta on generoitava jokaisella uudella pelikerralla satunnaisesti numeerisesta ’siemenestä’, joka mahdollistaa sen perusteella saman kartan generoinnin myös uudelleen. Kartalla liikkuminen on oltava vuoropohjaista. Kartan on koostuttava erilaisista maastotyypeistä, joilla on oltava vaikutusta pelaajan liikkumiseen. Karttapalat voivat sisältää vierailtavia kohteita, esineitä tai hirviöitä. Taistelun on oltava vuoropohjaista ja toimintopisteisiin nojaava. Taistelussa vihollisia voi tulla vastaan useissa ”aalloissa”. Pelaajat voivat taistella toisiaan vastaan. Varustuksen lisäksi pelaajan hahmon ominaisuuksia on voitava kehittää pelin edetessä. Varusteita, rahaa ja kokemusta on saatava taistelusta, joita pelaajan on voitava käyttää esim. kaupungista löytyvien kauppiaiden kanssa toimiessa tai suoraan hahmonsa parantelun. Pelaajille on annettava jokin päämäärä, mikä tulee saavuttaa, jotta pelin voi voittaa.

3.2 Käytetyt tekniikat

Projekti toteutettiin Javalla, koska Java on muodostumassa yhä suositummaksi kieleksi pelin kehityksessä, etenkin itsenäisten ja pienten kehittäjien keskuudessa. Javaan päätyminen johtui myös siitä, että se oli tutuin ohjelmointikieli meille molemmille. Javan peruskirjastoja käyttämällä emme kuitenkaan olisi päässeet toivomaamme lopputulokseen, joten niiden lisäksi valitsimme kirjastoja tietokantarakenteiden [28.], graafisen ulosannin, kuin myös ohjelmakoodin kääntämisen tekoon. Näiden lisäksi tarvitsimme keinot version- ja projektinhallintaan. Seuraavaksi esittelen lyhyesti projektin toteutuksessa käytettyjä tekniikoita ja sovelluksia.

3.2.1 Version One – projektinhallinta

Version One on työkalu, joka on suunniteltu tukemaan ketteriä ohjelmistokehitysmenetelmiä kuten Scrum (kuva 9). Selaimen kautta käytettävä ohjelma mahdollistaa projektin etenemisen yksityiskohtaisen seuraamisen ja hallinnan. Järjestelmään tallennetaan tuotteen kehitysjono, josta kootaan sprintit. Kehitysjonon tehtäville voidaan mm. määrittää tarkat kuvaukset, työmäärät, prioriteetit sekä kenen tehtäväksi se tulee. Ohjelma osaa myös luoda edistymiskäyrän kuvaamaan tuotteen tai sprintin jäljellä olevaa työmäärää ajan funktiona. [11.]

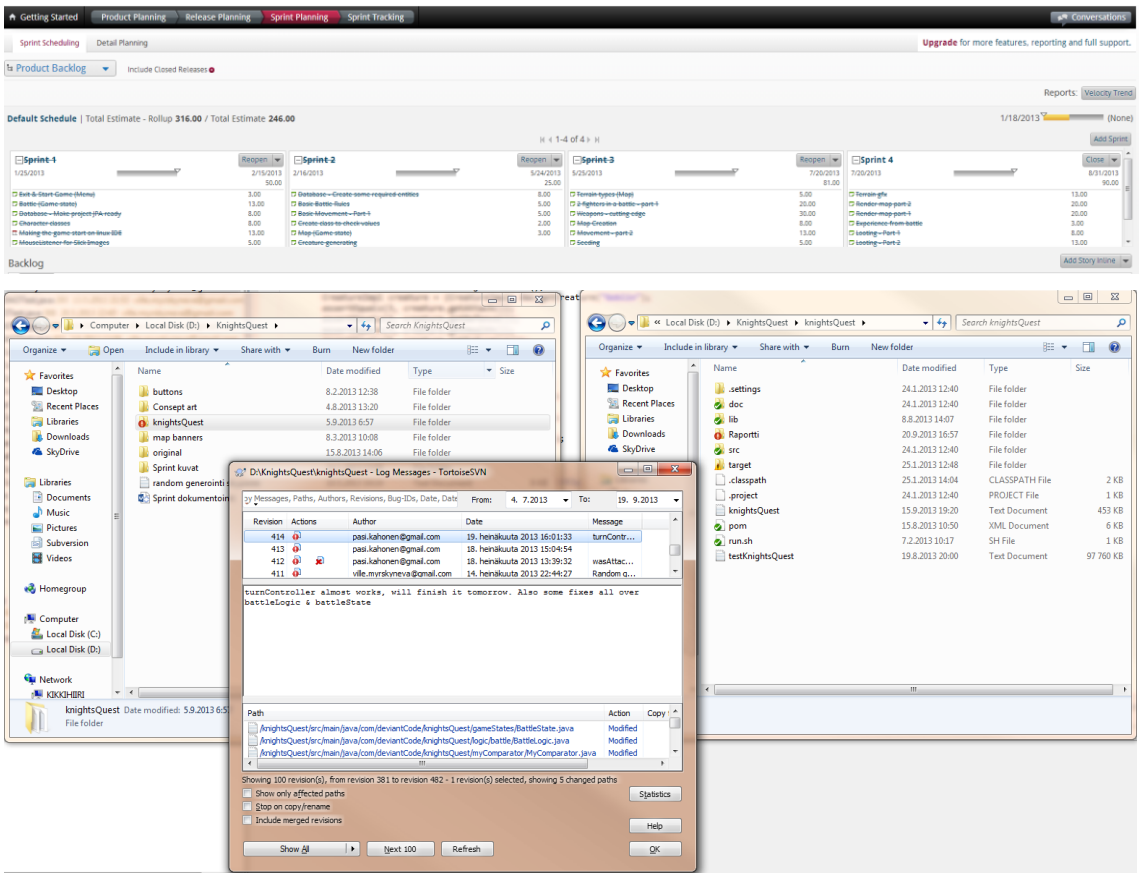


Kuva 9. VersionOnen etusivu

3.2.2 TortoiseSVN -versionhallinta

Tortoise SVN on ilmainen sovellus ohjelmistokehityksen versionhallintaan. Ohjelma on suunniteltu helpottamaan eri ohjelmaversioiden hallinnassa. Sovellus integroituu Windowsin resurssienhallintaan. Pienet kuvapäällysteet täydentävät tiedostojen varsinaisia kuvakkeita. Niiden ansiosta käyttäjä pystyy päättelemään suoraan työkopionsa tilan.

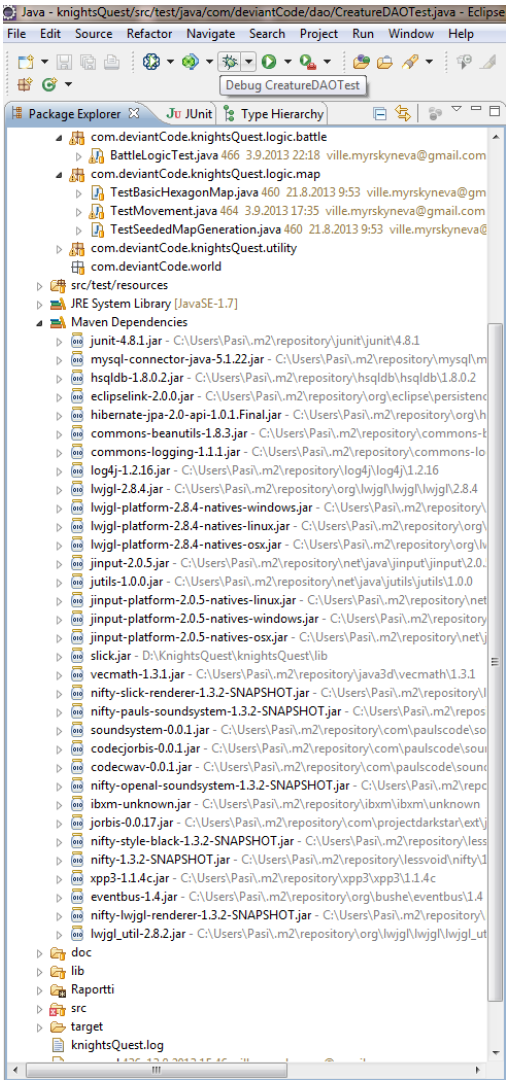
Graafinen käyttöliittymä mahdollistaa eri versioiden tutkimisen ja niihin liitettyjen kommenttien lukemisen (kuva 10). Muuttuneet tiedostot voidaan luetteloida ja yksittäiseen tiedostoon kohdistuneet muutokset ovat myös nähtävissä. Yksi sen tärkeimmistä ominaisuuksista on kuitenkin tiedostoristiriitojen havaitseminen. Tiedostoristiriita syntyy, kun kaksi tai useampi kehittäjää on muuttanut samoja rivejä samasta tiedostosta. Koska Subversion ei tiedä projektista mitään, se jättää ristiriitojen ratkaisemisen kehittäjän vastuulle. Käytännössä tämä tarkoittaa sitä, että kehittäjät hyväksyvät ristiriidassa olevista kohdista toisen. Tärkeimpänä ominaisuutena voidaan kuitenkin pitää sitä, että versionhallinnasta löytyy aina ajan tasalla oleva projektin lähdekoodit. Tällöin vältytään siltä, jos esimerkiksi yhden kehittäjän kovalevy hajoaa, voi hän kovalevyn vaihdettuaan ladata versionhallinnasta menetetyt lähdekoodit takaisin. Eclipse, ohjelmointiympäristöön [30.], ladattava lisäosa mahdollistaa suoraan kehitysympäristöstä tallentamisen versionhallinan säiliöön. [12; 13.]



Kuva 10. TortoiseSVN loki ikkuna sekä taustalla paikallinen kopio projektista

3.2.3 Maven

Maven on sovelluskehys ohjelmistoprojektin hallinnointiin. Maven mahdollistaa projektin rakentamisen/kääntämisen, versioinnin, dokumentoinnin ja riippuvuuksien hallinnoinnin. Koska Maven automatisoi edellä mainitut työt voi ohjelmoija keskittyä itse ohjelmoimiseen. Mavenia varten luodaan XML-tiedosto, joka kuvaa käännettävää sovellusta, sen riippuvuuksia eri moduuleihin ja komponentteihin, kääntöjärjestystä, kansioita sekä tarvittavia lisäosia. Maven lataa Javan kirjastot ja omat lisäosansa dynaamisesti Mavenin keskussäiliöstä ja tallentaa ne lokaalisti käyttäjän koneelle (kuva 11). [12. 13.]



Kuva 11. Mavenin hallinnoimat riippuvuudet

3.2.4 Lightweight Java Game Library & Slick2D

Lightweight Java Game Library, tai lyhyesti LWJGL, on avoimen lähdekoodin ohjelmakirjasto tietokonepelien kehittäjille. LWJGL yhdistää useita kirjastoja kuten OpenGL, OpenAL , OpenCL ja mahdollistaa erilaisten peliohjainten hyödyntämisen käyttöjärjestelmästä riippumatta. LWJGL:n päätarkoitus on tarjota Java-kehittäjille pääsy resursseihin, joita ei muuten olisi saatavilla tai olisivat muuten heikosti hyödynnettävissä nykyisellään Javassa. Käytännössä LWJGL tarjoaa käyttäjälleen kehittyneempiä grafiikan, äänen, rinnakkaisohjelmoinnin sekä ohjainkontrollien työstö välineet, kuin Java oletuskirjastoillaan. LWJGL on yleisesti käytettävä pohja Javalla toteutetuissa pelimoottoreissa ja apukirjastoissa (kuva 12).

Kuva 12. Noin 33 miljoonaa kopiota tähän mennessä myynyt Minecraft on toteutettu hyödyntämällä LWJGL-kirjastoja [27.]

Slick2D on LWJGL:n ympärille luotu työkalu ja apuvälinekirjasto, joka on tarkoitettu etenkin kaksiulotteisten pelien kehittämisen helpottamiseksi. Slick2D sisältää tuen muun muassa kuville, animaatiolle, partikkeleille, äänelle ja musiikille. Slick2D antaa täten valmiit työkalut pelinkehittäjälle pelinsä luomiseen ja nopeasti pääsyn sisällön tuottamiseen. [16; 17.]

3.3 Scrum ja Knights Quest

Knights Quest toteutettiin Scrumilla, jotta veljeni saisi menetelmästä lisää kokemusta toimimesta scrum-masterina (Certified Scrum Master 2012) ja jotta menetelmä tulisi itsellenikin tutuksi ja saisin siitä lisäksi hyvin materiaalia päättötyöhöni. Alusta asti oli selvää, että joudumme soveltamaan Scrumin toimintamalleja johtuen siitä, että toimimme itse sekä tuoteomistajana ja kehitystiiminä, jonka lisäksi veljeni toimisi scrum-masterina. Seuraava osio käsittelee tarkemmin näitä sovellutuksia ja sitä, kuinka projekti käytännössä eteni.

3.3.1 Palaverit

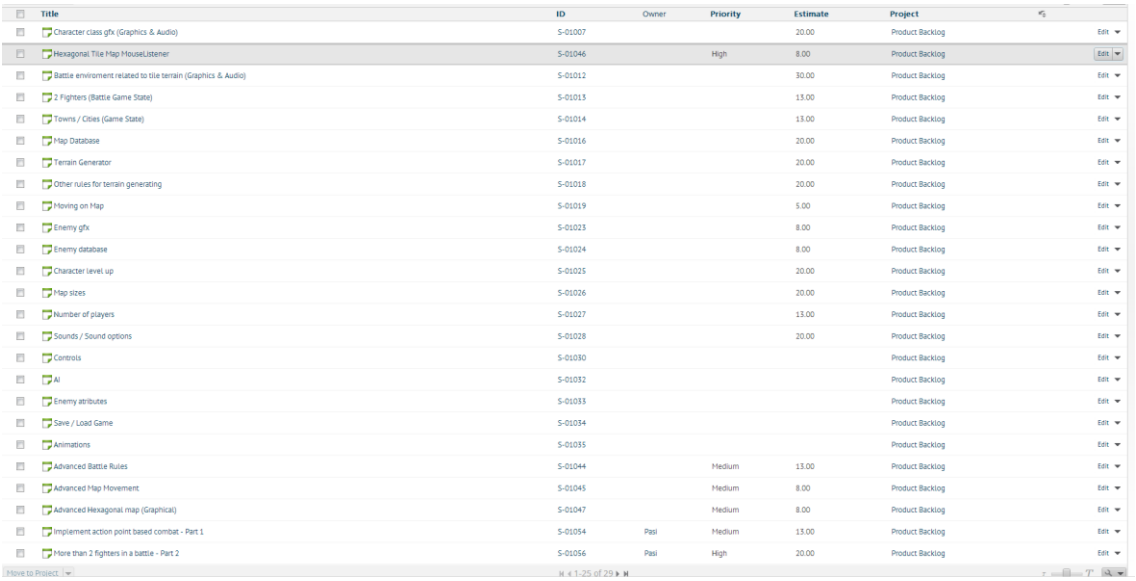
Ensimmäisessä suunnitelupalaverissa loimme pohjan tuotteen kehitysjonolle. Aloitimme kirjaamalla Post-It -lapuille peliin liittyviä teknillisiä vaatimuksia, logiikkaa ja sääntöjä. Suurimmille lapuille keräsimme hyvin pelkistettyjä vaatimuksia, kuten ”Vuoropohjainen” tai ”Tietomalli”. Näiden yläkäsitteiden alle kasasimme pienemmille lapuille tarkempia määritelmiä, joista nämä yläkäsitteet koostuvat. Tämän jälkeen aloitimme tuotejonon kohtien työmäärän arvioinnin.

Työmäärän arviointi tapahtui ensimmäisessä suunnittelupalaverissa ns. -suunnittelupokeri menetelmällä (kuva 13). Menetelmä on yleisesti käytössä ketterissä menetelmissä. Suunnittelupokeriin tarvitaan erikseen sitä varten suunniteltu korttipakka ja lista asioista joiden työmäärää arvioidaan. Käytännössä tämä tarkoitti, että otimme kehitysjonosta kohdan, keskustelimme lyhyesti sen sisällöstä, toteutuksesta ja riskeistä. Tämän jälkeen otimme molemmat omasta pakasta mielestämme työmäärää kuvaavan kortin ja asetimme sen pöydälle lukupuoli alaspäin. Kun molemmat olivat asettaneet kortit, käännettiin ne yhtäaikaisesti. Jos luvut täsmäsivät, siirryttiin arvioimaan seuraavaa kohtaa. Lukujen poiketessa paljon toisistaan molemmat perustelivat arvionsa ja tämän jälkeen, joko hyväksyimme jommankumman arviosta tai valitsimme kompromissiarvon näiden väliltä, jos vain oli mahdollista. Työmäärältään liian suuret tehtävät pyrimme pilkkomaan pienempiin osiin. [10.]

Kuva 13. Suunnittelupokerikortit [10.]

Koska emme tehneet projektia täysipäiväisesti, emme voineet pitää säännöllisesti päiväpalavereita. Pysyäksemme perillä toistemme tekemisistä, pyrimme raportoimaan edistymisestä tai ongelmista aina kuin mahdollista. Käytännössä tämä tarkoitti lähinnä etäviestintää, kuten tekstiviestejä, puheluita tai kommunikointia Skypen välityksellä. Näiden lisäksi huomasimme keskustelevamme projektista myös yhteisten harrastusten tai jopa illanistujaisten yhteydessä.

Ensimmäisen sprintin valmistuttua pidimme sulautetun palaverin, joka sisäisti sekä sprintti katselmuksen, retrospektiivin, kuin myös seuraavan sprintin suunnittelupalaverin. Palaverin aluksi esittelimme toisillemme miten olimme sprinttiin kuuluvat tehtävämme toteuttaneet. Todettuamme tehtävät ”valmiiksi” hyväksyimme ne ja lopuksi suljimme sprintin, tai jos huomasimme puutteita, lisäsimme sen seuraavaan sprinttiin. Tämän jälkeen aloimme työstää tuotteen kehitysjonoa (kuva 14), tarkensimme vaatimuksia ja lisäsimme uusia ja tarkistimme vanhoja tehtäviä, purimme liian suuria osia pienemmiksi tai poistimme turhia. Työmäärät arvioimme jälleen suunnittelupokerimenetelmää hyödyntäen. Lopuksi valitsimme mielestämme oleellisimmat tehtävät seuraavaa sprinttiä varten. Pidimme jatkossa edellä mainitun kaltaisen palaverin aina sprintin päätteeksi.

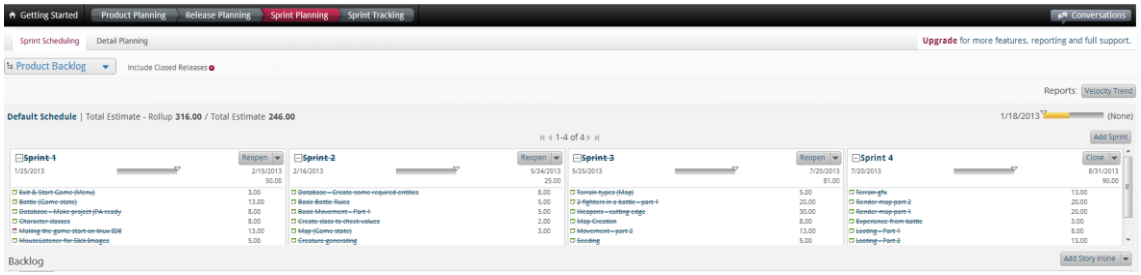


Title	ID	Owner	Priority	Estimate	Project
Character class gfx (Graphics & Audio)	S-01007			20.00	Product Backlog
Hexagonal Tile Map MouseListener	S-01046		High	8.00	Product Backlog
Battle enviroment related to tile terrain (Graphics & Audio)	S-01012			30.00	Product Backlog
2 Fighters (Battle Game State)	S-01013			13.00	Product Backlog
Towns / Cities (Game State)	S-01014			13.00	Product Backlog
Map Database	S-01016			20.00	Product Backlog
Terrain Generator	S-01017			20.00	Product Backlog
Other rules for terrain generating	S-01018			20.00	Product Backlog
Moving on Map	S-01019			5.00	Product Backlog
Enemy gfx	S-01023			8.00	Product Backlog
Enemy database	S-01024			8.00	Product Backlog
Character level up	S-01025			20.00	Product Backlog
Map sizes	S-01026			20.00	Product Backlog
Number of players	S-01027			13.00	Product Backlog
Sounds / Sound options	S-01028			20.00	Product Backlog
Controls	S-01030				Product Backlog
AI	S-01032				Product Backlog
Enemy atributes	S-01033				Product Backlog
Save / Load Game	S-01034				Product Backlog
Animations	S-01035				Product Backlog
Advanced Battle Rules	S-01044		Medium	13.00	Product Backlog
Advanced Map Movement	S-01045		Medium	8.00	Product Backlog
Advanced Hexagonal map (Graphical)	S-01047		Medium	8.00	Product Backlog
Implement action point based combat - Part 1	S-01054	Pasi	Medium	13.00	Product Backlog
More than 2 fighters in a battle - Part 2	S-01056	Pasi	High	20.00	Product Backlog

Kuva 14. Osa tuotteen kehitysjonosta

3.3.2 Sprintit

Vaihtelevista aikatauluista johtuen sprinttien pituuksien ja työmäärien pitäminen samoina oli täysin mahdotonta. Sprinttien pituudet vaihtelivat kolmen viikon pikasprintistä massiiviseen 3 kuukauteen. Samasta syystä myös työmäärät vaihtelivat runsaasti. Vaikka pisin sprintti oli pituudeltaan 3 kuukautta, oli se työmäärältään vähäisin, kun taas toiseksi lyhin sprinttimme, pituudeltaan noin 5 viikkoa, oli työmäärältään suurin. Sprinttien epätasaisuudesta huolimatta jokaisen sprintin lopussa oli toimiva tuoteversio, jonka ominaisuudet olivat kehittyneet edelliseen sprinttiin verrattuna (kuva 15).



Kuva 15. Valmistuneet sprintit

Ensimmäinen sprintti koostui seuraavista tehtävistä

- Exit & Start Game (Menu / Game State)
- Battle (Game State)
- Database Make Project JPA ready
- Character classes
- MouseListener for Slick Images.

Näistä tehtävistä kaksi ensimmäistä oli itselleni suunnattuja. Exit & Start Game -tehtävän määritelmänä oli, että peli käynnistettäessä aukeaa ikkuna. Ikkunassa on voitava navigoida näppäimistöllä tai hiirellä. On mahdollistettava Exit game, eli pelin sammuttaminen. Start game -valinnalla peli käynnistää Battle Staten. Battle Statelle riitti tässä vaiheessa, että pelitila on luotu ja se on käynnistettävissä aloitusruudun menun kautta, joka avaa päälle mustan ruudun. Työmääräni kuulostaa vähäiseltä, mutta näiden lisäksi käytin aikaa opiskellessani LWJGL:n ja Slick2d:n ominaisuuksia. Taistelutilaan siirryttiin lopulta enteriä painamalla, peli sammui esc -painikkeella.

Toinen sprintti koostui seuraavista tehtävistä

- Database Create some required entities
- Basic Battle Rules
- Basic Movement part 1
- Create class to check values
- Map (Game state)
- Creature generating
- Random number generator.

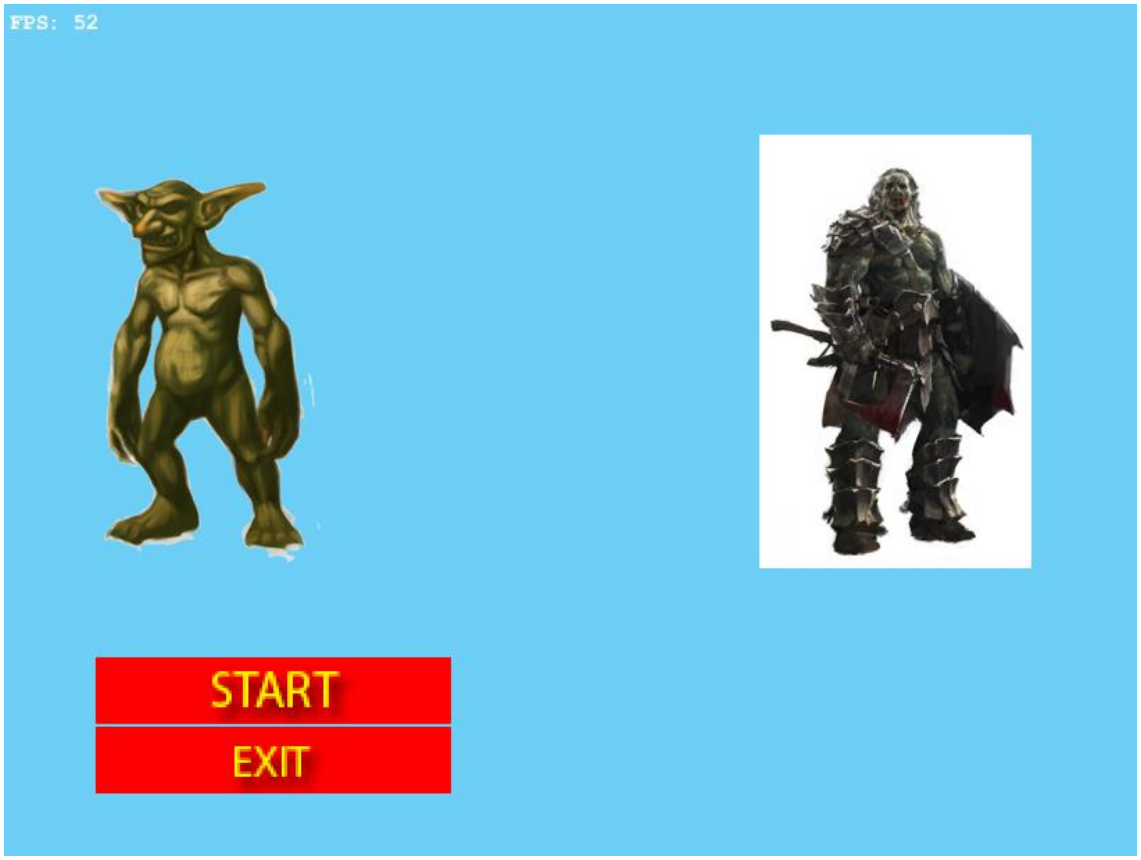
Näistä tehtävistä ainoastaan Map, eli karttapelitilan luominen oli vastuullani, sillä olin sprintin aikana häämatkallani, joten en voinut osallistua toteutukseen.

Kolmannen sprintin tehtäviä olivat

- Terrain types (Map)
- 2 fighters in a battle part 1
- Weapons cutting edge
- Map creation
- Movement part 2
- Seeding.

Näistä tehtävistä toteutin 2 fighters in a battle part 1:n sekä Weapons cutting edgen. Ensimmäinen mainituista tehtävistäni määriteltiin, että taistelussa on oltava vähintään 2 osapuolta. Luotiin käyttöliittymä toiminnoille, kuten hyökkäys, mikä kuluttaa toimintapisteitä. Keston mennessä nolliin on luotava ilmoitus, että olento on kuollut. Tehtävät olivat hyvin työläitä, ja suurin osa ajastani meni taistelulogiikan työstämiseen. Taistelulogiikan käsittämiä ominaisuuksia ovat mm. hyökkääminen, puolustaminen / vuoronlopettaminen, taistelusta pakeneminen, hahmon kuolema, taistelun vuoropohjaisuuden toteuttaminen sekä taistelijoiden järjestäminen alussa sen mukaan

kenellä on eniten toimintapisteitä. Puolustuksen ja taistelusta pakenemisen toteuttaminen siirtyivät ongelmien takia seuraavalle sprintille. Lisäksi näille kaikille toiminnoille oli luotava yksikkötestit joilla voitiin todentaa ohjelmallisesti tehtyjen toimintojen logiikan toimivuus. Taistelunäkymään kohdistunut työ piti sisällään hahmojen tyyppiä vastaavan kuvan hakemisen, sekä hiirtä tottelevien näppäimien tekemisen hyökkäykselle ja puolustukselle (kuva 16). Weapons cutting edge toimii pohjana aseille pelissä ja mahdollistaa teräaseiden käytön. Aseiden ominaisuudet ladataan erillisestä ominaisuustiedostosta, missä määritellään mm. aseen tyyppi, pohja vahinko, mahdollinen erikoisominaisuus, onko se yksi vai kaksi kätinen, paino, onko tehostettu taialla, aseen nimi ja sekä mahdollinen tarina aseen takana.

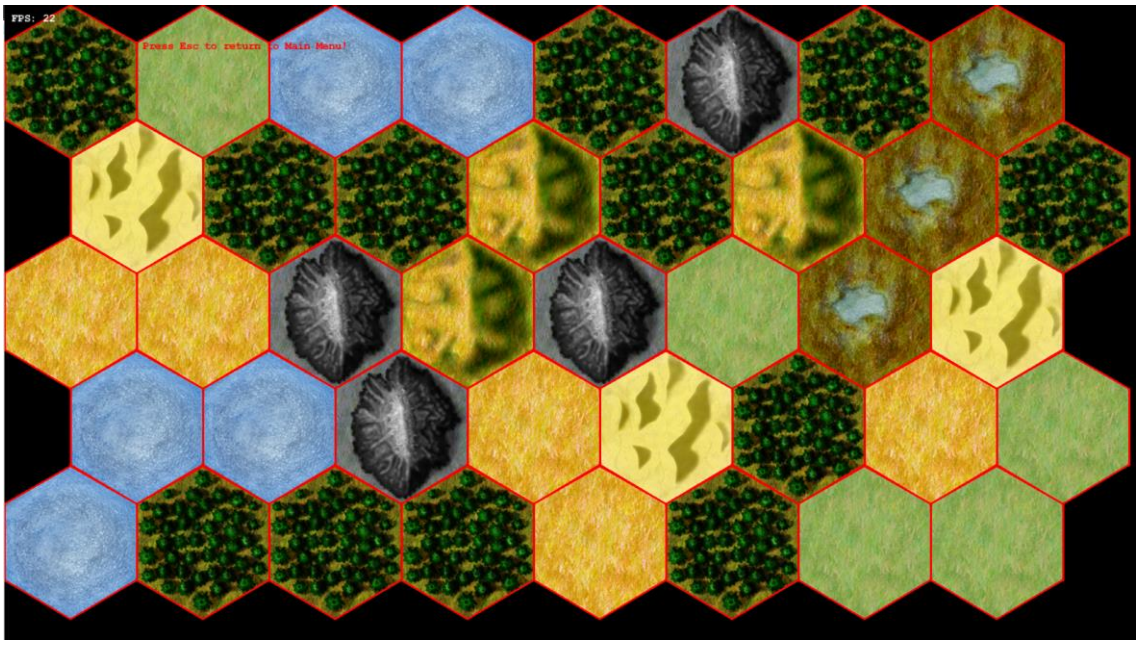


Kuva 16. ”Goblin”- ja ”Orc”- taistelutilan varhaisessa vaiheessa, menu -painikkeet start ja exit toimivat place holdereina hyökkäykselle ja vuoron lopettamiselle. [25; 26.]

Neljäs ja lopputyön kannalta viimein sprintti koostui seuraavista tehtävistä

- Terrain GFX
- Render map part 2
- Battle Part 2
- Render map part 1
- Experience from battle
- Looting – Part 1
- Looting – Part 2
- Movement part 3.

Tehtävistä kolme ensimmäistä olivat vastuullani. Terrain GFX -tehtävä tarkoitti piirtopöydän ja sen mukana tulleen Photoshop Elements 10:n opiskelua sekä lopulta kahdeksan erilaisen kuusikulmaisen karttapalan käsin piirtämistä. Piirtäminen oli mukavaa vaihtelua ja opiskellessa ja laattoja piirtäessä meni arviolta 7 työpäivää. Maastotyypeiksi valikoitui ruoho, aro, suo, vesi, kukkulat, vuori, metsä sekä aavikko. Render map part 2:ssa toteutin veljeni toteuttaman part 1:n pohjilta lopullisen kartan piirtämisen ja lomittamisen ruudulle (kuva 17). Battle part 2 käsitti edellisestä sprintistä toteuttamatta jääneet ylimääräisten toimintopisteiden käyttäminen vuoronlopussa puolustukseen sekä taistelusta pakeneminen. Pakenemisella oli lisäedellytyksenä, että pakenijalla on oltava täydet toimintapisteet pakoa yrittäessään, sekä vihollistaistelijoiden on saatava toimia kertaalleen pakenemistoiminnon aktivoinnin jälkeen.



Kuva 17. Render map part 2:n luoma kuva satunnaisgeneroidusta kartasta

Jokainen sprintti käsitti lisäksi kattavat testit toteutetuista toiminnoista. Viides ja sitä seuraavat sprintit tullaan toteuttamaan kehitysjonoon (liite 1) jääneiden toimintojen pohjalta.

3.3.3 Taistelulogiikan läpikäyntiä testien kautta

Projekti eteni testien ehdoilla. Saatuamme jotain mielestämme toimimaan, oli se testattava läpikotaisin Javan JUnit-testauskehystä hyödyntäen [29.] ennen kuin saattoi jatkaa seuraavaan tehtävään. Tässä osiossa käymme läpi pääosin tekemääni taistelulogiikkaan liittyviä testejä. Esittelen ensin testikoodin, jonka jälkeen käyn läpi, mitä testissä pääasiallisesti tapahtuu.

```
@RunsWith(JUnit4.class)
public class BattleLogicTest {
    Creature orc = null, rat = null;
    private static Logger log = Logger.getLogger(BattleLogicTest.class);
    private LinkedList<Creature> creatureList = null;

    private static final String SKELETON = "Skeleton";

    @Before
    public void setUp() {

        this.orc = CreatureDAO.getInstance().getCreature("Orc");
        this.rat = CreatureDAO.getInstance().getCreature("Rat");
        this.orc.setWieldedWeapon(WeaponDAO.getInstance().
        getWeapon("Katana"));
        this.rat.setWieldedWeapon(null);
        this.orc.setCurrentActionPoints(this.orc.getActionPoints());
        this.rat.setCurrentActionPoints(this.rat.getActionPoints());

        this.creatureList = new LinkedList<Creature>();
        this.creatureList.add(this.orc);
        this.creatureList.add(this.rat);

    }
```

Ensimmäinen rivi määrittää millä JUnitin versiolla testi ajetaan ja toinen rivi käsittää ainoastaan julkisen testiluokan nimen. Seuraavaksi esitellään kaksi Creature -luokan muuttujaa ”orc” ja ”rat”, jotka alustetaan tyhjiksi vielä tässä vaiheessa. Tämän jälkeen luodaan testiluokan tapahtumia tarvittaessa kirjaava lokikirja ”log”. Seuraavalla rivillä luodaan tyhjä linkitettylista ”creatureList”, johon tallennetaan Creature -tyyppisiä olioita. Tämän jälkeen luodaan merkkijono ”SKELETON”, johon tallennetaan sana ”Skeleton” Tätä käytetään myöhemmässä vaiheessa. Ylläolevasta koodista suoritetaan aina ensimmäisenä testiluokkaa ajettaessa @before merkinnän alle jäävä osio. Tässä luodaan kaksi hirviötä ”Orc” sekä ”Rat”. ”Orc” saa kannettavakseen Katana-miekan, ”Rat” ei saa mitään. Seuraavaksi molemmille otuksille asetetaan toimintapisteet, minkä rajoissa he voivat omalla vuorollansa taistelussa toimia. Viimeisellä kolmella rivillä luodaan varsinainen linkitetty lista Creatureille, johon ”Orc” ja ”Rat” lisätään.

```
  @Test
    public void testWasAttackSuccessful() {

        try {
            assertTrue(BattleLogic.wasAttackSuccessful(orc, 8, rat, 2));
        } catch (Exception e) {
            assertFalse(e.getMessage(), true);
        }
    }

    @Test
    public void testUnsuccessfulAttack() {
        boolean attackOutcome;

        try {
            attackOutcome = BattleLogic.wasAttackSuccessful(this.orc, 3,
this.rat, 4);
            assertFalse("Attack was suppose to be unsuccessful", attackOut-
come);
        } catch (Exception e) {
            assertFalse(e.getMessage(), true);
        }
    }

    @Test
    public void testUnsuccessfulAttackWithEqualDieRolls() {
        boolean attackOutcome;

        try {
            attackOutcome = BattleLogic.wasAttackSuccessful(this.orc, 3,
this.rat, 5);
            assertFalse("Attack was suppose to be unsuccessful", attackOut-
come);
        } catch (Exception e) {
            assertFalse(e.getMessage(), true);
        }


    }
```

Edellä olevissa kolmessa @Test -merkinnän alla olevissa testeissä testataan hyökkäyslogiikkaa. Hyökkäys onnistuu, jos hyökkäävän otuksen hyökkäysarvo on suurempi kuin puolustavan puolustusarvo. Metodiin wasAttackSuccessful() annetaan parametreina hyökkäävä olio, sen hyökkäyspisteet, puolustava olio ja sen puolustuspisteet. Normaalitilanteessa arvot tulisivat satunnaisgeneraattorin arpomina jotka huomioivat hyökkäykseen ja puolustukseen liittyviä muuttujia, mutta testitilanteessa arvot on syötettävä käsin, jotta voidaan olla varmoja lopputuloksesta. Ensimmäisen testin pitäisi antaa onnistunut hyökkäys, hyökkäyksen ollessa 8 ja puolustuksen 2. Seuraavien kahden pitäisi palauttaa epäonnistunut hyökkäys, keskimmäisessä hyökkäysarvon ollessa pienempi, 3, kuin puolustus, 4, ja viimeisessä niiden ollessa yhtä suuret, eli 3.

```
@Test
    public void testDamageDealt() {

        try {
            Creature skeleton = null;
            skeleton = CreatureDAO.getInstance().getCreature(SKELETON);
            int currentHealt = skeleton.getCurrentHealth();
            assertTrue("Healt should be atleast 2", currentHealt >= 2);
            boolean damageDealt = BattleLogic.dealDamage(1, this.orc, skele-
ton, 5, 3);

            assertTrue("Was suppose to deal damage succesfully", damage-
Dealt);
        } catch (DeathException death) {
            assertFalse("The creature was not suppose to die: " + death.get-
Message(), true);
        }


    }
```

Tämä testi on käytännössä suoraa jatkoa edellisille. Testi tarkistaa, toimiiko vahingon jakaminen oikein onnistuneen hyökkäyksen jälkeen. Testissä ”Orc” saa vastaansa ”Skeleton” Creaturen, joka luodaan, minkä jälkeen tarkistetaan, että ”Skeleton” omaa vähintään 2 terveyspistettä. Tämän jälkeen totuusarvoja sisältävään ”damageDealt” -muuttujaan asetetaan dealDamage -metodin palauttama, true- tai false-arvo. Metodiin asetetaan ensimmäiseksi arvo vähittäisvahingolle, sitten hyökkäävä ja puolustava olio ja lopuksi hyökkäys- ja puolustuspisteet. Lopuksi tarkistetaan, palauttaako toiminto oikean arvon. Mikäli vahinkoa aiheutettiin enemmän kuin oli tarkoitus, palauttaa testi tapauksesta virheilmoituksen.

```
@Test
    public void testCurrentDefence() {
        orc.setCurrentDefence(orc.getDefence());
        assertEquals(4, orc.getCurrentDefence());

    }

    @Test
    public void testUseDefence() throws IllegalValueException {
        creatureList.peekFirst().setCurrentDefence(creature-
List.peekFirst().getDefence());
        orc.setCurrentActionPoints(orc.getActionPoints());
        BattleLogic.useDefence(creatureList);

        assertEquals(9, creatureList.peekLast().getCurrentDefence());
    }

    @Test
    public void noApToUseDefence() throws IllegalValueException {
```

```
        creatureList.peekFirst().setCurrentDefence(creature-
List.peekFirst().getDefence());
        creatureList.peekFirst().setCurrentActionPoints(0);
        BattleLogic.useDefence(creatureList);
        assertEquals(4, creatureList.peekLast().getCurrentDefence());

    }

    @Test
    public void testUseDefenceInCombat() {
        try {
            creatureList.peekFirst().setCurrentDefence(creature-
List.peekFirst().getDefence());
            creatureList.peekFirst().setCurrentActionPoints(creature-
List.peekFirst().getActionPoints());
            BattleLogic.useDefence(creatureList);
            assertEquals(9, creatureList.peekLast().getCurrentDefence());

            creatureList.peekFirst().setCurrentActionPoints(creature-
List.peekFirst().getActionPoints());
            assertEquals(9, creatureList.peekLast().getCurrentDefence());
            assertFalse(BattleLogic.wasAttackSuccessful(creature-
List.peekFirst(), 5, creatureList.peekFirst(),
                    creatureList.peekLast().getCurrentDefence()));
        } catch (Exception e) {
            assertFalse(e.getMessage(), true);
        }


    }
```

Edellä olevat 4 testiä käyvät läpi Creatureiden puolustukseen liittyviä toimintoja. Ensimmäinen testi yksinkertaisesti tarkistaa, että ”Orc” saa hahmolomakkeestaan oikeat puolustuspisteet ja vertaa sitä oletettuihin pisteisiin. Seuraava testi tarkistaa, että kun ”Orc” käyttää puolustuksensa parantamiseen kaikki vuoronsa toimintopisteet, tämän puolustus on 9. Kolmas testi tarkistaa, että puolustukseen ei mene bonuspisteitä, jos toimintopisteitä ei ole käytettäväksi vuoronlopussa. Viimeisessä testissä kokeillaan puolustusbonuksen käyttöä taistelussa. Taisteluvuorossa oleva olio käyttää puolustustoimintoaan heti vuoron alussa, jolloin kaikki toimintopisteet käytetään puolustuksen pohja-arvon nostattamiseen. Tämän jälkeen tapahtuu vuoronvaihto ja seuraavana vuorossa oleva hyökkää juuri puolustusta käyttäneen kimppuun arvolla 5, jolloin hyökkäyksen tulisi epäonnistua.

```
@Test(expected = UnableToComplyException.class)
    public void testCanNotPerformAttack() throws UnableToComplyException {
        // The "Goblin" is needed to have an image
        Player player = new Player("Sir DeathTest", 0, "Goblin");
        Weapon weapon = null;
        try {
            weapon = WeaponDAO.getInstance().getWeapon("Rapier");
        } catch (Exception e) {
            assertFalse(e.getMessage(), true);
        }
        int ap = weapon.getWeaponNeededActionPoints();
        assertTrue(ap > 1);
        player.setWieldedWeapon(weapon);
        this.creatureList.peekFirst().setCurrentActionPoints(1);
        try {
            BattleLogic.wasAttackSuccessful(player, 10, this.creature-
List.peekFirst(), 0);
        } catch (IllegalValueException ive) {
            assertFalse(ive.getMessage(), true);
        }


    }
```

Tässä testissä luodaan Player-luokan olio, player, jolle annetaan parametreina, nimi, pelaajanumero sekä kuva, joka on tässä tapauksessa ”Goblin”. Testin tarkoituksena on varmistaa, että hyökkääminen ei ole mahdollista, jos käytetyn aseen vaatima toimintopisteiden määrä ylittää hahmon käytettävissä olevat. Luodaan weapon-olio ja muuttujaan ap tallennetaan miekan vaatimat toimintapisteet. Tämän jälkeen tarkistetaan, että toimintapisteitä tarvitaan aseen käyttöön enemmän kuin 1. Tämän jälkeen ase annetaan playerille. Playerin toimintapisteet vähennetään yhteen pisteeseen, jonka jälkeen kutsutaan wasAttackSuccesful -metodia, toiminnon pitäisi epäonnistua, koska pelaaja ei voi käyttää asetta, jos toimintapisteitä ei ole riittävästi.

```
  @Test
    public void getNextInTurnAndEscapingForMultipleCreaturesTest() {
        CreatureDAO creatureDao = CreatureDAO.getInstance();
        LinkedList<Creature> creatureList = new LinkedList<Creature>();
        Creature goblin = creatureDao.getCreature("Goblin");
        Creature skeleton = creatureDao.getCreature("Skeleton");

        this.orc.setCurrentActionPoints(2);
        this.orc.setTurnEnded(true);
        creatureList.add(this.orc);

        creatureList.add(goblin);

        this.rat.setEscapeMode(true);
        creatureList.add(this.rat);

        creatureList.add(skeleton);
```

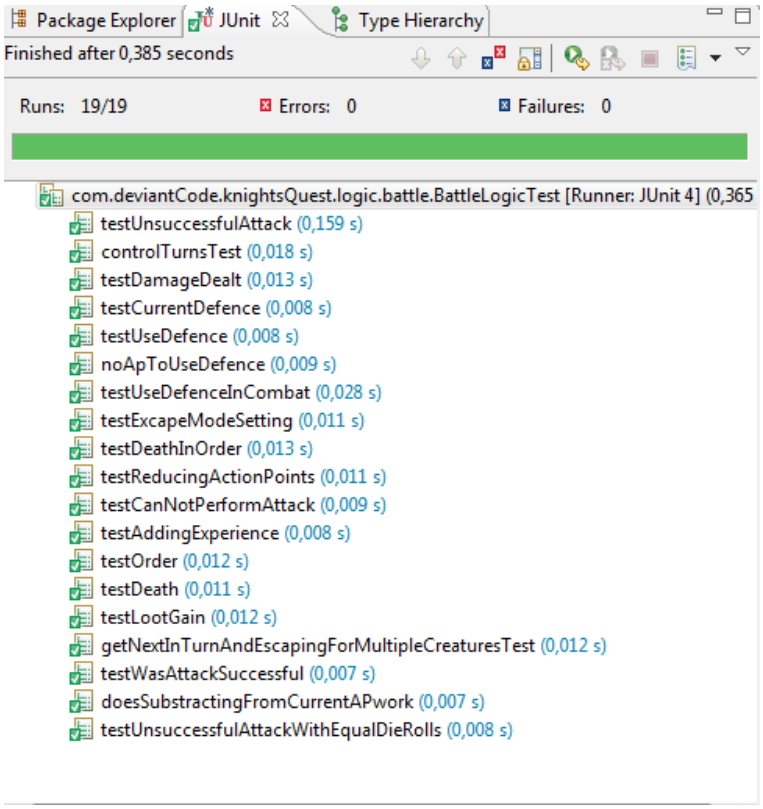
```
        assertEquals(4, creatureList.size());
        assertEquals(this.orc, creatureList.peekFirst());
        Creature goblinNext = BattleLogic.getNextInTurn(creatureList);
        assertEquals(goblin, goblinNext);

        goblinNext.setCurrentActionPoints(0);

        // The rat should escape now
        Creature skeletonNext = BattleLogic.getNextInTurn(creatureList);
        assertEquals(skeleton, skeletonNext);
        assertEquals(3, creatureList.size());
        assertFalse("The rat should had escaped", creatureList.con-
tains(this.rat));

    }
```

Edellä oleva testi tarkistaa, että vuoronvaihto ja pakeneminen toimii taistelussa, jossa on useampi Creature. Testin alussa luodaan uusi linkitettylista sekä kaksi uutta Creature luokan olioita ”Goblin” ja ”Skeleton”. Tämän jälkeen jo aiemmin luodulle ”Orc”:lle asetetaan toimintapisteet kahteen ja käsketään lopettamaan vuoronsa, jolloin puolustuspisteet nousevat myös jäljelle jääneiden toimintapisteiden verran. ”Orc” lisätään testin alussa luotuun creatureList listaan ”Goblin” kanssa. Seuraavaksi ”Orc” kanssa samaan aikaan luotu ”Rat” asetetaan pakenemaan, jonka jälkeen se lisätään creatureListin jatkoksi ”Skeleton” perässä. Tarkistetaan listan koko, jonka pitäisi olla 4. Tarkistetaan, että listan ensimmäisenä on ”Orc”. Luodaan uusi Creature -luokan olio goblinNext, johon tallennetaan seuraavana listassa vuorossa oleva Creature. Tarkistetaan, onko goblin ja goblinNext samat oliot. Asetetaan goblinNextin toimintopisteet nollaan. Luodaan skeletonNext Creature johon tallennetaan jälleen listalta seuraava. Muuttujien skeleton ja skeletonNext tulisi olla samat. Lista, mihin Creaturet oli tallennettu, tulisi olla nyt yhden pienempi, eli 3, rotan paettua onnistuneesti. Kuvassa 18 näkyy onnistuneesti suoritetut taistelulogiikkatestit.



Kuva 18. Onnistuneesti ajetut taistelulogiikkatestit

3.4 Mietteitä Scrumista

Aikataulun ja pienen kehitystiimin tuottamista haasteista ja veljeni opastuksen ansiosta uskon sisäistäneeni Scrumin opit vähintäänkin tyydyttävästi. Uskoisin Scrumin olevan käytännön työelämässä huomattavasti helpompaa ja pystyväni jo työskentelemään osana scrum-tiimiä.

Täysipäiväisesti työskennellessä etuna on päivittäinen työskentely kehitystiimin kanssa, joka pystyy tukemaan toisiaan heti paikan päällä. Tietenkin sain veljeltäni tukea ja neuvoja aina tarvittaessa, mutta lähinnä puhelimessa tai tietokoneelta hoidettu kommunikointi hankaloittaa aina asioita, kun ei voi suoraan näyttää ongelman aiheuttajaa. Projektia täysipäiväisesti työstettäessä työhön on helpompi päästä uudelleen sisälle, kun työpäivien väliin jää korkeintaan viikonloppu. Vaihtelevasta aikataulusta johtuen Knights Questin työstöpäiville saattoi kertyä eroa jopa lähes kuukauden verran. Välin kasvaessa työpäivien välillä oli yhä vaikeampaa muistaa, mitä oli tehnyt ja mihin jäänyt, lähdekoodin kommenteista huolimatta. Kolmantena helpottavana asiana, omaan scrum-työskente-

lyymme poiketen, täysipäiväisessä projektin toteutuksessa on isoimpien palaverien pitäminen erillään, mikä mahdollistaa täten tarkemman tuotejonon hallinnan ja sprintin tehtävien määrittelyn.

Vaikka projekti ei edennyt täysin scrumin teorioiden mukaan ja törmäsimme matkalla tästä johtuviin ongelmiin, tiedän sen silti soveltuvan loistavasti pelinkehitykseen, sillä useat suuret pelialan vaikuttajat kuten Ubisoft, CCP ja Blizzard käyttävät scrum-viitekehystä tuotteidensa teossa. Vaikka emme seuranneetkaan täydellisesti scrumin ohjeita, niin kyseessähän on todellakin vain suuntaa antava viitekehys, jota pyrimme toteuttamaan parhaamme mukaan. Aiomme jatkaa samaa kaavaa hyödyntäen, jossa tavoitteena on lopulta peli, jota voimme ylpeänä kutsua omaksemme.

4 Yhteenveto

Opinnäytetyössä tavoitteena oli tutustua Scrum-sovelluskehitysprosessiin sekä soveltaa sitä pelinkehityksessä. Tämän lisäksi työssä esiteltiin vielä toistaiseksi keskeneräiseksi jäänyt Knights Quest -peli, kuinka Scrumia sovellettiin sitä tehdessä sekä lyhyesti pelinteossa käytettyjä tekniikoita.

Scrumin teorian sisäistäminen oli helppoa, kiitos Internetistä löytyvän kattavan dokumentaation ja osaavan scrum-masterin. Etenkin jälkimmäisen sekä scrumin viitteellisyyden ansiosta menetelmän soveltaminen oli suhteellisen helppoa, scrum-tiimin pienestä koosta ja vaihtelevista työajoista huolimatta. Menetelmä osoittautui toimivaksi malliksi pelinkehitykseen tuoden projektiin joustavuutta ja selkeyttä. Scrumin toimivuutta pelinkehityksessä tukee myös tieto siitä, että se on laajasti käytössä pelialan yrityksissä.

Knights Quest -projekti saavutti sille asettamamme tavoitteet ennalta asetettuun päivämäärään mennessä. Oli tietoinen ratkaisu kehittää pääasiallisesti pelin tietomallia ja taustalogiikkaa ja jättää suurin osa audiovisuaalisesta työstä myöhemmäksi. Tämä mahdollisti selkeämmän kuvan saamisen ohjelman rakentumisesta sekä tarjosi paremmat mahdollisuudet yksikkötestauksen oppimiselle. Koska toteutuksessa oli myös muita intressejä, kuin pelkän graafisen ulkoasun tekeminen, ei graafista valmistumista pidetty oleellisena tällä aikataululla.

Työn painopisteenä oli scrum kehitysmallin, sekä teknisiin aspekteihin tutustuminen. Kehityksen kannalta tässä vaiheessa oli kannattavinta luoda toimiva logiikka ja tietomalli, joka tukee graafista kehitystä ilman että on tarvetta toteuttaa ”tilapäisratkaisuja” käyttöliittymän toimintoja tukemaan. Pitkällä tähtäimellä ennalta toteutettu ja testattu toimiva tietomalli ja logiikka ovat kehityksen kannalta nopeampi tapa edistää sovellusta. Tämä on kuitenkin hieman scrumin periaatteen vastaista, koska sen perusajatus on toteuttaa nopeasti toiminnallista näyttöä.

Knights Questin kehitys tulee jatkumaan. Jatkossa kehitys tulee keskittymään yhä enemmän käyttäjälle suoraan näkyvään toimintaan ja audiovisuaaliseen antiin.

Lähteet

1 Scrum (FI). Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Scrum>. Luettu 7.9.2013

2 The Scrum Guide. Verkkodokumentti. Scrum.org: Ken Schwaber ja Jeff Sutherland <https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum%20Guide%20-%20FI.pdf#zoom=100>. Luettu 7.9.2013

3 Ketterä ohjelmistonkehitys. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Ketter%C3%A4_ohjelmistokehitys>. Luettu 7.9.2013

4 Ketterän ohjelmistokehityksen julistus. Verkkodokumentti. Agile Manifesto. <http://agilemanifesto.org/iso/fi/>

5 Hirotaka Takeuchi & Ikujiro Nonaka. 1986 .New New Product Development Game.

6 Scrum (EN). Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Scrum_(software_development)>. Luettu 7.9.2013

7 Ketteryys Haltuun: Scrum pähkinänkuoressa. Verkkodokumentti. Sininen meteoriitti. <http://www.meteoriitti.com/Artikkelisarjat/Ketteryys-haltuun/Ketteryys-haltuun-Scrum-pahkinankuoressa/>. Luettu 7.9.2013

8 Catanin Uudisasukkaat. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Catanin_uudisasukkaat>. Luettu 8.9.2013

9 Moonstone: A Hard Days Knight. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Moonstone:_A_Hard_Days_Knight>. Luettu 8.9.2013

10 Suunnittelupokeri. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Suunnittelupokeri>. Luettu 8.9.2013

11 VersionOne: Agile Project Management Tool Overview. Verkkodokumentti. <http://www.versionone.com/product/agile-project-management-tool-overview/>. 13.9.2013.

12 TortoiseSVN. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/TortoiseSVN>. Luettu 13.9.2013)

13 TortoiseSVN-ohjelman ominaisuudet. Verkkodokumentti. <http://tortoisesvn.net/docs/nightly/TortoiseSVN_fi/tsvn-preface-features.html>. Luettu 13.9.2013

14 Mikä on Maven. Verkkodokumentti. Joona Viertola. <http://www.sci-onar.com/mika-on-maven/>. Luettu 13.9.2013

15 Apache Maven. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Apache_Maven>. Luettu 13.9.2013

16 Lightweight Java Game Library. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/LWJGL>. Luettu 13.9.2013

17 Slick2D Java Game Library. Verkkodokumentti. <http://slick.ninjacave.com/wiki/index.php?title=Main_Page>. Luettu 13.9.2013

18 Incremental Iterative. Kuva. <http://www.planetgeek.ch/wp-content/uploads/2010/09/4agile.png>. Katsottu 13.9.2013

19 Scrum Roles. Kuva. <http://www.microtool.de/instep/en/images/scrum/das-ist-Scrum_1.png>. Katsottu 13.9.2013

20 Product Owner. Kuva. <http://noopera-tion.typepad.com/.a/6a00e54ff8b9c188340133f5604563970b-200pi>. Katsottu 13.9.2013

21 Scrum Master. Kuva. <http://qeworks.com/wp-content/uploads/2013/05/TheScrumMaster.jpg>. Katsottu 13.9.2013

22 Scrum-prosessi. Kuva. <http://upload.wikimedia.org/wikipedia/commons/5/58/Scrum_process.svg>. Katsottu 13.9.2013

23 Tehtävälista. Kuva.<http://www.thematrixfiles.net/wp-content/uploads/2009/08/office-not-done.jpg>. Katsottu 13.9.2013

24 Valmiin määritelmä. Kuva. <http://2.bp.blogspot.com/_Sj9LfQtNy3s/SglGuBp-rfI/AAAAAAAAABA/gEPAHG3lNTs/s400/Done.jpg>. Katsottu 13.9.2013

25 Goblin. Kuva. <http://images3.wikia.nocoo-kie.net/__cb20130310164244/weirdcommunity/images/b/bc/11-5-goblin_color1.jpg>. Katsottu 13.9.2013

26 Orc. Kuva. <http://stuffershack.com/wp-content/uploads/2011/06/orc.jpg>. Katsottu 13.9.2013

27 Minecraft. Kuva. <http://screenshots.nl.sftcdn.net/nl/scrn/6649000/6649795/world-of-warcraft-for-minecraft-05-700x407.jpg>. Katsottu 13.9.2013

28 Java Persistence Api. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Java_Persistence_API>. Luettu 23.9.2013

29 JUnit. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/JUnit>. Luettu 23.9.2013

30 Eclipse IDE. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Eclipse_ide>. Luettu 23.9.2013

Knights Quest - kehitysjono

Title	Priority	Esti-mate
Battle enviroment related to tile terrain (Graphics & Audio)		30,00
Character class gfx (Graphics & Audio)		20,00
Map Database		20,00
Terrain Generator		20,00
Other rules for terrain generating		20,00
Character level up		20,00
Map sizes		20,00
Sounds / Sound options		20,00
More than 2 fighters in a battle - Part 2	High	20,00
2 Fighters (Battle Game State)		13,00
Towns / Cities (Game State)		13,00
Number of players		13,00
Advanced Battle Rules	Me-dium	13,00
Implement action point based combat - Part 1	Me-dium	13,00
Hexagonal Tile Map MouseListener	High	8,00
Enemy gfx		8,00
Enemy database		8,00
Advanced Map Movement	Me-dium	8,00
Advanced Hexagonal map (Graphical)	Me-dium	8,00
Moving on Map		5,00
Weight limit exceed affect to endurance on map move-ment	Me-dium	5,00
Movement part 4	Low	3,00
Controls		
AI		
Enemy atributes		
Save / Load Game		
Animations		
Battle Part 3	Low	

Creatures UML-kaavio