Hirvonen Tuomas

Ketterät kehitysmenetelmät ja työkalut Internet-palvelun uudistusprojektissa

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Mediatekniikan koulutusohjelma
Insinöörityö
5.12.2011

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Tuomas Hirvonen Ketterät menetelmät ja työkalut Internet-palvelun uudistusprojektissa 32 sivua + 2 liitettä 7.11.2011
Tutkinto	insinööri (AMK)
Koulutusohjelma	mediatekniikka
Suuntautumisvaihtoehto	digitaalinen media
Ohjaajat	ohjelmistokehittäjä Tommi Rautanen lehtori Ilkka Kylmäniemi

Insinöörityön aiheena oli MTV3:n Katsomo-verkkopalvelun uudistusprojekti. Palvelu uudistettiin käyttöliittymän ja joidenkin ominaisuuksien osalta. Insinöörityössä paneuduttiin uudistusprojektissa käytettyihin projektinhallintamenetelmiin sekä työkalujen käyttöön. Projektinhallintamenetelmänä käytettiin Scrumia ja sovelluksen kehityksessä hyödynnettiin prototyyppejä käyttäviä kehitysmenetelmiä. Projektinhallinnan työkaluina käytettiin JIRA- sekä Basecamp-tehtävänhallintajärjestelmiä.

Projektissa oli mukana neljä yritystä. Palvelun konsepti ja käyttöliittymän suunnittelu toteutettiin Palmu Inc. muotoilutoimistossa. Taustajärjestelmään tehtävät muutokset teetettiin Norjassa Vimond-yhtiössä. Cybercom Plenware Oy:ltä oli kolme ohjelmistokehittäjää mukana toteutuksessa ja MTV3:lta oli mukana projektipäällikkö sekä yksi ohjelmistokehittäjä. Neljän yrityksen yhteistyö toi erityisiä haasteita projektinhallinnan, kommunikoinnin ja työprosessien suhteen.

Projektissa sovellettiin vahvasti ketterää Scrum-projektinhallintamenetelmää. Scrumin käyttö toi projektiin selkeyttä ja joustavuutta, mutta ei kuitenkaan kankeita dokumentaatioita tai prosesseja. Myös muutoksiin oli helppo reagoida ja mahdolliset ongelmat havaittiin hyvissä ajoin, jolloin ne pystyttiin ratkaisemaan helposti. Scrumin lyhyet iteraatiot ja päivittäiset kokoukset toivat projektiin hyvän tahdin ja pitivät huolen että työt edistyvät. Ongelmia projektinhallinnassa ilmeni Cybercomin ja taustajärjestelmästä vastaavan yrityksen välillä. Osa ongelmista oltaisiin voitu ratkaista yhtenäistämällä yritysten projektinhallintaa ja käyttämällä yhteistä Scrum-prosessia.

Koska sovelluksen taustajärjestelmään tehtävät muutokset toteutettiin toisessa yrityksessä, käytettiin kehityksen apuna prototyyppejä. Käyttöliittymästä luotiin prototyypit, jotka toimitettiin määrittelyn ohella taustajärjestelmästä vastanneelle yritykselle.

Tehtävänhallinnan työkaluna projektissa käytettiin JIRA- sekä Basecamp-tehtävänhallintajärjestelmiä. Näistä JIRA osoittautui erittäin käteväksi työkaluksi Scrum-projektin hallinnassa.

Avainsanat	ketterä ohjelmistokehitys, Scrum, projektinhallinta, prototyypit, tehtävänhallinta

Metropolia

Abstract

Author Title Number of Pages Date	Tuomas Hirvonen Agile methodologies and tools in website renewal project 32 pages + 2 appendices 7 November 2011
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	Tommi Rautanen, Software Designer Ilkka Kylmäniemi, Lecturer

The purpose of the present thesis was renewing of the Katsomo web site. The whole user interface and some of the main functionalities were renewed in the project. This thesis takes a closer look on an agile project management methodology Scrum and prototype driven software development. Task management tools JIRA and Basecamp were used in conjunction with these methodologies and those tools are also covered in the thesis.

There were four companies participating in this project. The concept and user interface were designed by Palmu Inc. design office. Changes to the backed system were implemented in Norway by Vimond. Three software designers were participating from Cybercom Plenware Oy. MTV3 acted as a customer and project manager and one software designer was from there. Cooperation of the four companies brought some challenges related to project management, communication and working processes.

The Scrum methodology was utilized strongly in this project. Scrum brought flexibility and clarity to the project and cut down unnecessary documentation and processes. Adapting to changes was easier because of the use of Scrum. Also, upcoming problems were noticed earlier so they were easier to resolve. Use of short iterations and daily Scrum meetings helped to keep within the time schedule. Some problems occurred in project management which were related to cooperation between us and the Norwegian company. Some problems could have been resolved by unifying our project management and using only one Scrum process.

Because the changes to backend system were implemented in the other company, we used prototypes to aid the development. We implemented prototypes of the user interface and backend changes were implemented based on the prototypes and written specifications.

We used JIRA and Basecamp issue tracking systems for task management. JIRA was a helpful tool because it has very good support for managing Scrum projects.

Keywords	agile software development, project management, Scrum, prototypes, issue tracking

Sisällys

Lyhenteet, käsitteet ja määritelmät

AJAX	*Asynchronous Javascript And XML*. tekniikka asynkroniseen tiedon välittämiseen selaimen ja palvelimen välillä.
CSS	*Cascading Style Sheets*. HTML-dokumenttien tyylien määrittelemiseen käytetty kieli.
HTML	*Hypertext Markup Language*. kuvauskieli, jota käytetään Internet-sivujen rakenteen kuvaamiseen.
JavaScript	Ohjelmointikieli, jota Internet-selaimet käyttävät.
JBoss	Avoimeen lähdekoodiin perustuva sovelluspalvelin Java-sovelluksien ajamiseen.
JSON	*JavaScript Object Notation*. tiedonsiirtoon käytettävä yksinkertainen rakennekieli.
NITF	*News Industry Text Format*. standardoitu XML-määritys tekstipohjaisten uutisartikkeleiden kuvaamiseen.
VMAN	*Video Man*. MTV3:n videomateriaalin hallintaan käyttämä järjestelmä.
XML	*Extensible Markup Language*. kuvauskieli tiedon merkityksen sekä rakenteen kuvaamiseen.
XSLT	*Extensible Stylesheet Language Transformations*. XML-pohjainen merkintäkieli, jota käytetään XML-dokumenttien muuttamiseen toisenlaiseen muotoon.

1 Johdanto

Insinöörityön aiheena on MTV3:n Katsomo-verkkopalvelun uudistusprojekti. Insinöörityö tehdään Cybercom Plenware Oy:lle, joka on noin 400 henkeä Suomessa työllistävä ohjelmistonkehityspalveluita tuottava yritys. Sen asiakkaisiin kuuluu Suomen suurimpia media-alan yrityksiä, kuten MTV3, Alma Media ja YLE.

Katsomo on tv-ohjelmien suoratoistoa tarjoava verkkopalvelu. Nykyisessä palvelussa voi katsoa useita MTV3-kanavan ohjelmia sekä ohjelmiin liittyvää lisämateriaalia, jota ei televisiossa näytetä. Katsomossa näytetään myös suorana lähetyksenä esimerkiksi urheilua ja uutisia. Se on yksi Suomen suurimpia netti-tv-palveluita.

Vanha Katsomo uudistetaan palvelukonseptin, ulkoasun, käyttöliittymän ja monien toiminnallisuuksien osalta. Eri työvaiheet tulevat jakaantumaan useamman yrityksen kesken, jolloin voidaan puhua hajautetusta ohjelmistoprojektista. Uudistetun palvelukonseptin, graafisen ilmeen ja sivuston käyttöliittymän on suunnitellut Palmu Inc. muotoilutoimisto. Taustajärjestelmiin tehtävät muutokset teetetään Norjassa Vimond-nimisessä TV-alan yrityksessä, jolta MTV3:n emoyhtiö tilaa ohjelmistoihin liittyviä palveluita.

Cybercomin tehtäväksi jää projektin käytännön toteuttaminen eli konseptisuunnitelman ja rautalankamallien muuttaminen toimivaksi sovellukseksi. Projektipäällikkö on MTV3:lta kuten myös yksi ohjelmistokehittäjäkin.

Tässä työssä keskitytään ohjelmistokehitysprojektin hallintaan liittyvien menetelmien ja työkalujen käyttöön. Tarkemmin paneudutaan ketterään Scrum-projektinhallintamenetelmään, Rapid Application Prototyping -kehitysmalliin ja JIRA- sekä Basecamp-tehtävänhallintajärjestelmien hyödyntämiseen edellä mainittujen menetelmien käytössä. Kaikki nämä ovat olennaisessa asemassa tämän projektin läpi viemisessä ja tavoitteiden saavuttamisessa. Aiheita ei käydä täydellisesti läpi, vaan esitellään tämän projektin kannalta tärkeimmät asiat.

2 Katsomo-verkkopalvelun uudistus

Yleistä Katsomon uudistuksesta

Katsomo-videopalvelu uudistetaan palvelukonseptin, ulkoasun, käyttöliittymän ja joidenkin toiminnallisuuksien osalta. Palvelun tekninen alusta säilyy samana.

Uudistusprojektin on tarkoitus kestää noin yhdeksän kuukautta ja ensimmäinen versio sivustosta on tarkoitus julkaista noin kolmen kuukauden kuluttua projektin alkamisesta. Tämä versio sisältää kaikki vaatimusmäärittelyssä keskeiseksi palvelun käytön kannalta asetetut ominaisuudet. Näihin kuuluvat videoiden toisto, tuotteiden ostaminen, käyttäjätietojen hallinta, ohjelmien omat sivut, kategoriasivut, hakutoiminnallisuus, videolistaukset ja ohjelmalistaukset. Projektin loppuaika käytetään vähemmän tärkeiden ominaisuuksien toteuttamiseen, testaukseen ja virheiden korjaamiseen. Vähemmän tärkeät ominaisuudet sisältävät muun muassa erilaisia kommentointimahdollisuuksia, videoiden soittolistan ja käyttäjien pisteytysjärjestelmän. [1.]

Lähtökohta ja tavoitteet

Lähtökohtana on iso uudistusprojekti, jossa on mukana useita yrityksiä ja aktiivisesti noin kymmenen henkilöä. Projektilla on kohtalaisen tiukka aikataulu ja sovittuna julkaisupäivänä pitäisi pystyä julkaisemaan toimiva sivusto, jossa on tärkeimmät ominaisuudet mukana. Projektin aikana täytyy varautua muutoksiin, kuten joidenkin toiminnallisuuksien pois jättämiseen tai prioriteettien muutoksiin. Sovellusta tulisi rakentaa siten, että siitä on olemassa jatkuvasti toimiva versio, jota voidaan esitellä projektiryhmälle kommentointia ja arviointia varten.

Eri yritysten rooli projektissa

Palvelukonseptin sekä ulkoasun on suunnitellut Palmu Inc. ja se on toimittanut projektia varten konseptisuunnitelman, käyttöliittymän rautalankamallit ja visuaaliset kuvat sivustosta, joiden pohjalta uudistus tehdään. Tarvittaessa Palmu Inc. konsultoi projektin edetessä mahdollisissa käyttöliittymään ja ulkoasuun liittyvissä asioissa. Kehitysryhmään kuuluu kolme henkilöä Cybercomilta ja yksi MTV3:n työntekijä. Projektipäällikkö on myös MTV3:lta. Kehitystiimin tehtävänä on teknisen toteutuksen suunnittelu ja

toteuttaminen. Sivuston taustajärjestelmästä vastaa norjalainen TV-alan yhtiö, joten taustajärjestelmään tehtävät muutokset teetetään siellä. Uusi sivusto valmistuu siis kolmen eri yrityksen yhteistyönä. Tämä tuo lisähaasteita projektiin, kuten kommunikointiin ja aikataulujen sovittamiseen yritysten välillä.

Työn haasteet

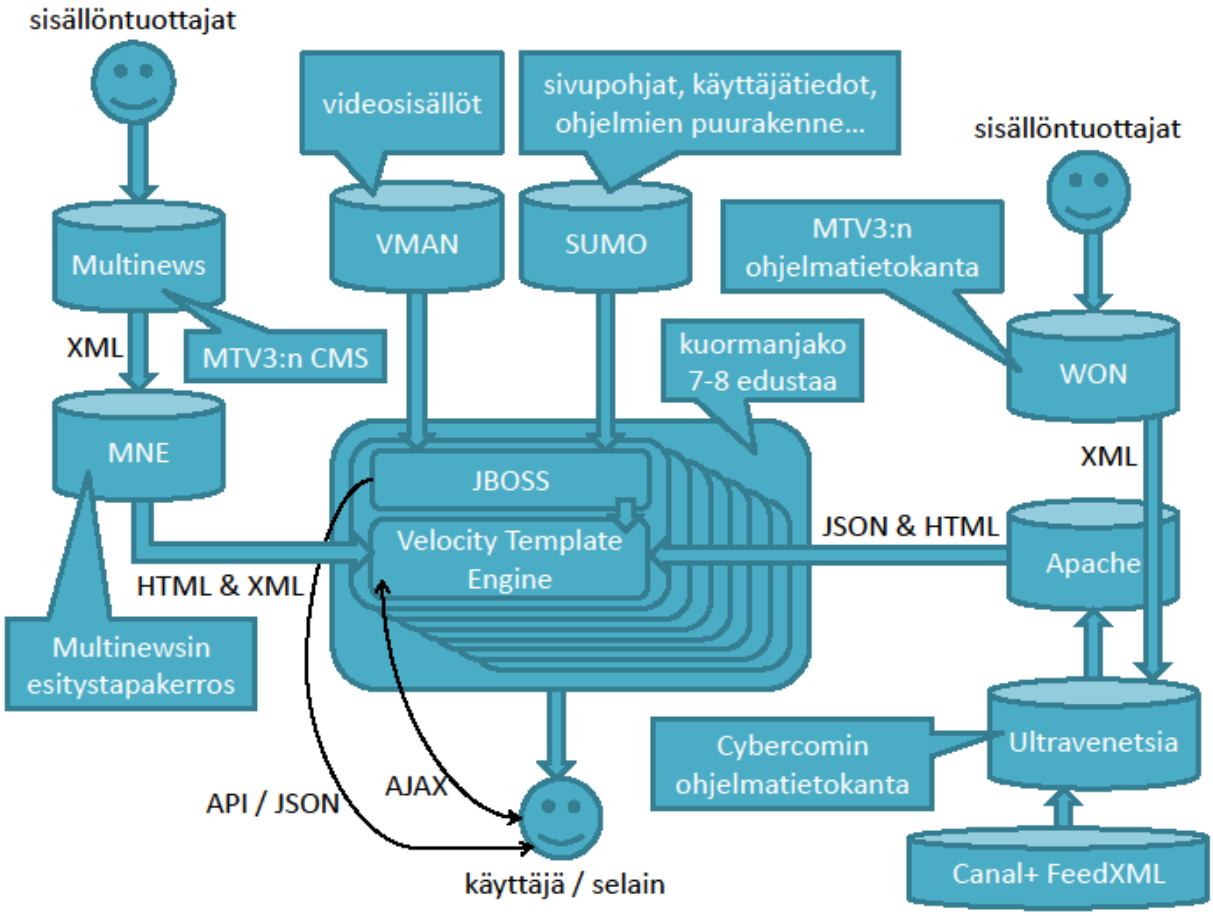
Yksi isoimmista haasteista tässä projektissa on järjestelmän eri osien kehityksen hajaantuneisuus. Taustajärjestelmän kehitys tapahtuu Norjassa eikä projektiryhmä pääse tekemään suoraan muutoksia sinne, vaan muutokset täytyy tilata erikseen norjalaisilta. Tähän haetaan helpotusta prototyyppien käytöstä, josta kerrotaan luvussa neljä. Myös sisältö tuodaan järjestelmään hyvin monesta eri lähteestä, kuten MTV3:n sisällönhallintajärjestelmästä ja ohjelmatietokannasta. Kun tällaiseen järjestelmään lähdetään tekemään muutoksia, on todennäköistä, että muutokset vaikuttavat hyvin moneen kytköksissä olevaan järjestelmään.

Projektin tehtävien ulkoistaminen toiseen maahaan tuo aina omat haasteensa. Välimatka, kieli ja mahdollinen aikaero hankaloittavat kommunikointia. Myös projektin aikataulut voivat olla vaikeita sovittaa kahden yrityksen välillä ja työt saattavat jäädä junnaamaan paikalleen jos toinen yritys ei saa tarvittavia tehtäviä valmiiksi aikatauluun mennessä. Myös kommunikointikanavien, kuten sähköpostin ja Skypen valinta ja käyttö, tuottavat haasteita.

Teknisen toteutuksen kannalta hankala asia on vaatimus joidenkin vanhojen tuotteiden säilyttämisestä ennallaan. Tämä tarkoittaa sitä, että tehtävät uudistukset eivät saa näkyä tietyillä sivuston osioilla ja vanhaa koodia joudutaan säästämään paljon. Tämä tulee aiheuttamaan ristiriitaisuuksia uuden ja vanhan koodin välillä. Vaarana on, että kun muutokset viedään julkaisun aikaan tuotantoon, vanhat tuotteet, joista käyttäjät ovat maksaneet, lakkaavat toimimasta. Tämän onnistuminen varmistetaan huolellisella testauksella. Projektissa toimii yksi henkilö testaajana, joka testaa kehityksen ohessa sovelluksen toimintoja sitä mukaa kuin ne valmistuvat.

Palvelun tekninen kuvaus

Katsomo-palvelua ajetaan JBoss-sovelluspalvelimella. Edustoja on käytössä kahdeksan, joista yksi toimii testiedustana, mutta se voidaan tarvittaessa valjastaa tuotantokäyttöön. Liikenne jaetaan palvelimien kesken kuormanjakajan avulla. Sovelluksen toiminnan kannalta keskeiset tiedot kuten käyttäjätiedot, sivupohjat, tuoterakenne ja ohjelmien puurakenne sijaitsevat SUMO-tietokannassa. Videosisällöt sijaitsevat VMAN-tietokannassa. Videosisältöjä jaellaan erillisiltä mediapalvelimilta suoraan käyttäjälle. JBossilla ajettavan sovelluksen ja edellä mainittujen tietokantojen kehitys tapahtuu Norjassa. Tietokantojen ylläpito on taas Cybercomin vastuulla. Kuvio 1 kuvaa Katsomon teknistä toteutusta ja kytköksiä ulkoisiin järjestelmiin. [2.]



Kuvio 1. Katsomon tekninen kuvaus ja kytkökset ulkoisiin järjestelmiin.

Sovelluksen esitystapakerroksessa käytetään Apache Velocity Template Engine -sivupohjamoottoria. Sovelluksen sivupohjat rakennetaan HTML-merkkikielellä ja taustasovelluksen välittämää tietoa tulostetaan HTML-koodin sekaan Velocity-syntaksin avulla. Sivupohjiin sisällytetään myös ulkoisista lähteistä tuotetut HTML-tiedostot.

Sivustolle tuotetaan sisältöä useista ulkoisista lähteistä, kuten MTV3:n sisällönhallintajärjestelmästä ja ohjelmatietokannasta. Multinews on MTV3:n sisällönhallintajärjestelmä. Multinewsilla tuotettu data tallennetaan NITF-standardin mukaisiin XML-tiedostoihin, joista tuotetaan edelleen sisältöä eri julkaisukanaviin MNE-esitystapakerroksen kautta. MNE käyttää sillä tuotettavan datan rakenteen määrittelyyn XSL-tiedostoja, joissa määritellään sivupohjat eri julkaisukanavia varten. MNE:llä voi julkaista dataa missä tahansa muodossa XSLT-muunnoksen avulla. Yleisimmin dataa julkaistaan HTML-, XML-, JSON- ja TXT-muodossa. MNE julkaisee myös kuvia ja liitetiedostoja. Katsomoon julkaistavat tiedostot kopioidaan suoraan Katsomon ensimmäiselle edustapalvelimelle, jonka kautta niitä jaellaan muille palvelimille. [3.]

Katsomoon tuodaan ohjelmatietoja Cybercomin hallinnoimasta UltraVenetsia-ohjelmatietokannasta. UltraVenetsia-tietokantaan taas tuodaan sisältöä MTV3:n omasta WON-ohjelmatietokannasta sekä ulkoisista lähteistä, kuten Canal+ FeedXML -palvelusta. WONista tuodaan tietoja UltraVenetsiaan Unix-palvelimella ajautuvien skriptien avulla. Näiden skriptien avulla myös UltraVenetsian tiedoista tuotetaan HTML- ja JSON-tiedostoja, jotka kopioituvat automaattisesti Katsomon ensimmäiselle edustapalvelimelle. [2.]

Katsomon kehitystä varten on erillinen kehitysympäristö, joka sijaitsee omalla palvelimellaan. Kehitystä varten on myös oma tietokanta. Kehityspuolelta valmista koodia viedään tuotantopalvelimille käsin kopioimalla, koska suurin osa koodista ei ole versionhallinnassa. Versionhallinnan kehittäminen Katsomon ympäristössä olisi varmasti jatkossa harkitsemisen arvoinen asia.

Oma rooli projektissa

Itse olen projektissa ohjelmistokehittäjän roolissa. Työtehtäviini kuuluu sivuston käyttöliittymän, sivupohjien ja Multinews-konfiguraatioiden toteuttamista. Käyttöliittymän

toteuttaminen sisältää HTML-rakenteen, CSS-tyylit ja JavaScript-toiminnallisuudet. Sivustolle rakennetaan paljon selaimessa tapahtuvia toimintoja sekä AJAX-toiminnallisuuksia, joten JavaScript-koodia täytyy kirjoittaa runsaasti. Luvussa 4.4 esitetyn videotaulukkokomponentin toteuttaminen oli yksi isompia tehtäviäni tässä projektissa.

Katsomon etsitystapakerros toteutetaan Velocity-sivupohjien avulla, joihin käyttöliittymän rakenne määritellään ja taustajärjestelmän palauttama data käsitellään ja esitetään käyttöliittymässä Velocity-koodin avulla.

Teen myös MTV3:n Multinews-sisällönhallintajärjestelmään tarvittavia muutoksia, jotta sen avulla pystyy tuottamaan uuden Katsomo-palvelun vaatimia sisältöjä. Muutokset toteutetaan määrittelemällä julkaistavan datan rakenne XSL-määrittelykielen avulla ja määrittelemällä tarvittavat julkaisuun liittyvät asetukset XML:n avulla.

Ohjelmointityön lisäksi tehtäviini kuuluu myös taustajärjestelmien muutosten määrittelyä. Taustajärjestelmän muutokset tilataan alihankintana, mutta Cybercomin täytyy kuitenkin itse konseptin perusteella määritellä tarvittavat taustajärjestelmän muutokset ja uusien ominaisuuksien vaatimukset.

3 Projektinhallinta

3.1 Projektinhallinnan käytäntöä

Projekti on väliaikainen ponnistus jonkun tietyn tavoitteen saavuttamiseksi. Projekti eroaa tavallisesta tuotantotyöstä siten, että projektilla on aina alku ja loppu. [4.]

Projektinhallinnan tehtävänä on työn ja resurssien organisointi siten, että projekti saavuttaa sille asetetut vaatimukset aikataulun ja budjetin puitteissa. Ohjelmistonkehitysprojektissa resurssit ovat pääsääntöisesti ihmisiä, kun taas muissa projekteissa resursseilla voidaan tarkoittaa esimerkiksi raaka-aineita ja energiaa. [4.]

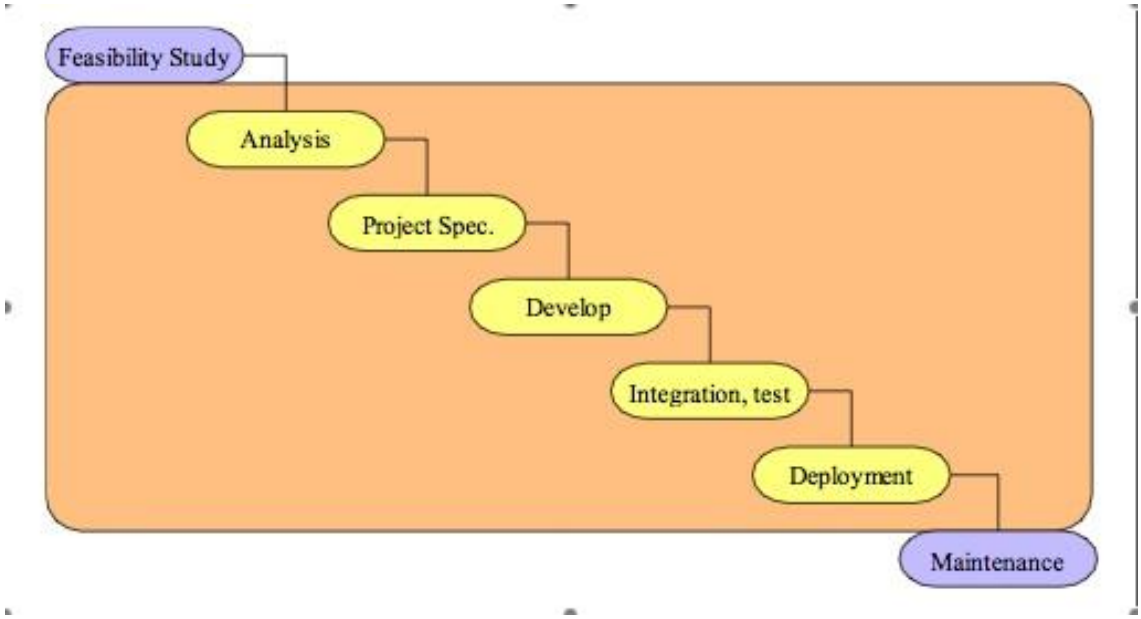
Projektinhallinnan isänä pidetään amerikkalaista insinööriä Henry Ganttia. Hän kehitti 1900-luvun alussa Ganttin kaavion, joka esittää projektin eri vaiheet visuaalisesti aika-

janalla. Tämä kaavio muistuttaa hyvin paljon vesiputousmallin toimintaa. Teollisuudessa projektinhallintaa alettiin hyödyntää systemaattisesti 1950-luvulla. [4.]

Oikean projektinhallintamenetelmän valintaan vaikuttavat projektin budjetti, projektiryhmän koko, käytettävä teknologia, käytettävät työkalut, projektin kriittisyys ja mahdolliset samanaikaiset projektit. [5.]

3.2 Perinteiset projektinhallintamenetelmät

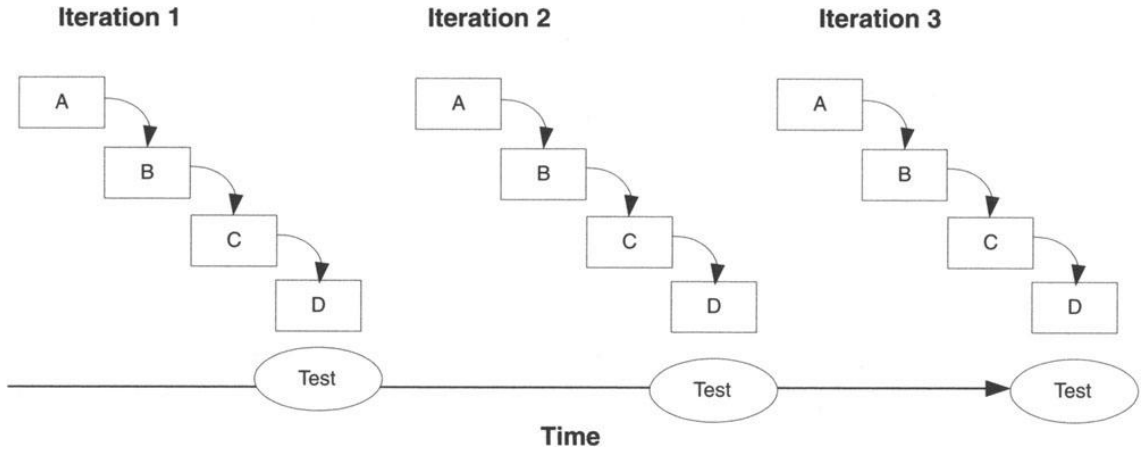
Perinteinen noin 1970-luvulta asti ohjelmistotuotannossa käytetty malli on vesiputousmalli. Vesiputousmallissa projekti muodostuu peräkkäisistä vaiheista, joista seuraavaan siirrytään vasta edellisen valmistuessa. Jokaisen vaiheen on tarkoitus tuottaa dokumentti, joka toimii pohjana seuraavalle vaiheelle. Vesiputousmalli sisältää yleensä ainakin seuraavat vaiheet: vaatimusmäärittely, toiminnallinen määrittely, toteuttaminen, testaaminen ja ylläpito. Kuvio 2 havainnollistaa vesiputousmallin etenemistä vaiheittain. [5.]



Kuvio 2. Vesiputousmalli ohjelmistoprojektissa [5].

Vesiputousmallia on myös mahdollista soveltaa iteratiivisesti. Tämä vähentää riskejä ja helpottaa muutosten hallintaa. Kuvio 3 havainnollistaa iteratiivista vesiputousmallia. Siinä projekti on jaettu kolmeen iteraatioon, joista jokaisessa toistetaan samat vaiheet alusta loppuun. Iteraation lopussa on olemassa versio tuotteesta, jota voidaan esitellä

asiakkaalle. Mahdolliset asiakkaan toivomat muutokset voidaan ottaa huomioon seuraavan iteraation määrittelyvaiheessa. [5.]



Kuvio 3. Vesiputousmalli iteratiivisesti sovellettuna [5].

Vesiputousmallin suurin heikkous on muutoksien ja riskien hallinta. Ohjelmistotuotantoprojektin alussa ei usein tiedetä kaikkia vaatimuksia, vaan ne muotoutuvat projektin edetessä. Perinteinen vesiputousmalli ei ota projektin aikana sen vaatimuksiin ja määrittelyyn tapahtuvia muutoksia ollenkaan huomioon. Vesiputousmallia käyttävässä projektissa muutosten toteuttaminen kesken projektin on usein aikaa vievää ja kallista. Kun esimerkiksi projektin toteutusvaiheessa halutaan tehdä muutoksia, täytyy palata ensimmäiseen määrittelyvaiheeseen ja käydä kaikki vaiheet uudestaan läpi muutosten osalta ja selvittää miten muutokset vaikuttavat sovelluksen muihin osiin. Toinen heikkous on myös dokumentoinnin suuri määrä. Dokumentointi vie aikaa ja saattaa tuottaa myös turhia dokumentteja. [5.]

Tyypillisesti vesiputousmallissa asiakkaan vaatimusmäärityksen perusteella tehdään projektin määritelmä, jonka asiakas hyväksyy. Projekti toteutetaan alussa tehdyn määritelmän mukaan, ja yleensä asiakas näkee tuotteen vasta valmiina projektin loputtua. Tällöin asiakkaalla ei ole mahdollisuutta päästä vaikuttamaan projektin määritelmään kesken projektin. Jos asiakas haluaa teettää muutoksia nähdessään valmiin tuotteen, on se yleensä kallista ja aikaa vievää. [5.]

Vesiputousmalli sopii parhaiten isoihin projekteihin, jotka ovat monimutkaisia mutta helposti ymmärrettävissä. Vesiputousmallia käytettäessä sovelluksen koodista tulee yleensä laadukasta ja helposti ylläpidettävää. Vesiputousmalli toimii hyvin, jos projektin

laatuvaatimukset menevät kustannus- ja aikatauluvaatimusten edelle. Web-ohjelmistojen kehitysprojekteissa näin ei kuitenkaan yleensä ole, vaan aikataulut ovat tiukkoja ja vaatimukset muuttuvat jatkuvasti. [6.]

3.3 Ketterä ohjelmistokehitys

Ketterän ohjelmistokehityksen periaatteita ovat ohjelmiston ensisijaisuus, suora viestintä ja nopea muutoksiin reagointi. Ketterät menetelmät korostavat ihmisten roolia ja yhteisöllisyyttä projektissa prosessien ja työkalujen sijaan. Toimiva ohjelmisto on ensisijainen tavoite ja dokumentaatiota, kuten diagrammeja ja kaavioita, pyritään karsimaan, koska ne eivät tuo lopulliselle tuotteelle mitään lisäarvoa. Asiakas on projektissa kokoajan läsnä ja asiakkaalla on mahdollisuus vaikuttaa projektin kulkuun. Sujuva kommunikaatio asiakkaan kanssa menee sopimusten tai tiukasti noudatettavien määrittelydokumenttien edelle. [7; 8.]

Ketterille kehitysmenetelmille yhteistä on se, että ne jakavat projektin lyhyisiin iteraatioihin, jotka kestävät yleensä kahdesta neljään viikkoon. Ohjelmistosta on tarkoitus julkaista toimivia versioita mahdollisimman usein. Ohjelmiston määrittelyä tehdään asteittain iteraatioiden kuluessa ja tämä mahdollistaa muutosten helpon tekemisen kesken projektin. Ketteriä menetelmiä käytettäessä ohjelmistoon tai sen vaatimuksiin kohdistuvat muutokset ovat helpommin hallittavissa kuin perinteisiä menetelmiä, kuten vesiputousmallia käytettäessä. [5.]

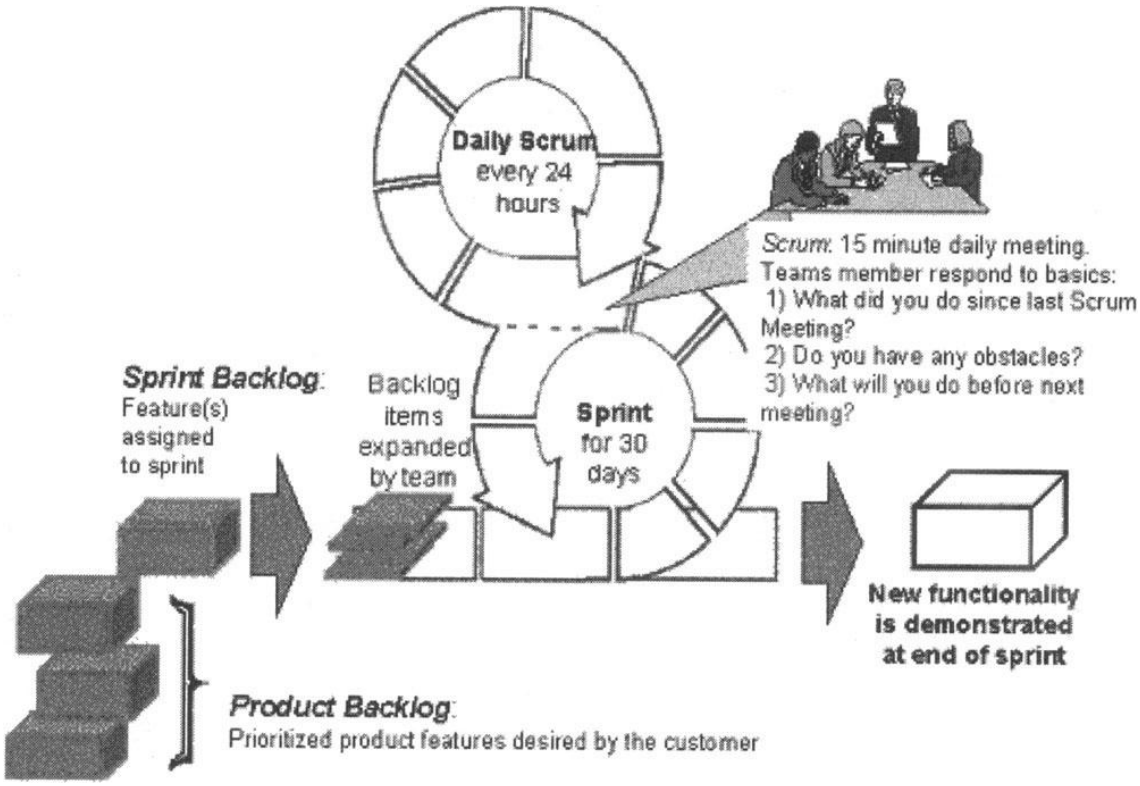
Yleisimmin käytettyjä ketteriä ohjelmistonkehitysmenetelmiä ovat Extreme Programming (XP), Scrum, Crystal methodology, Dynamic Systems Development Methodology (DSDM), Rapid Application Development (RAD), Adaptive software development, Lean development ja Feature-driven development. Kaikki edellä mainitut menetelmät sopivat parhaiten pienehköihin projekteihin, joissa työskennellään pienissä ryhmissä. Isoihin, pitkiin ja monimutkaisiin projekteihin sopivat paremmin perinteiset vesiputousmallin tyyppiset menetelmät. [5.]

3.4 Scrum-projektinhallintamenetelmä

Scrumin periaatteet

1980-luvulla Japanissa kehitettiin holistiseksi malliksi kutsuttu menetelmä, jota hyödynnettiin auto- ja elektroniikkateollisuudessa. Se perustui peräkkäisiin iteraatiohin, joiden aikana tuote valmistuu inkrementaalisesti kasvaen. 1990-luvun aikana holistinen menetelmä nimettiin Scrumiksi. Ensimmäinen Scrumia käsittelevä kirja: "Agile Software Development with Scrum" julkaistiin vuonna 2001. Ohjelmistokehityksessä Scrum on kohtalaisen tuore malli. [9.]

Scrumissa projekti jaetaan yleensä 30 päivän mittaisiin iteraatiohin eli pyrähdyksiin. Ohjelmisto on jaettu komponentteihin, jotka on jaettu pienempiin tehtäviin. Jokaiseen iteraatioon valitaan joukko tehtäviä, jotka kehitysryhmä uskoo saavansa valmiiksi iteraation aikana. Iteraation aikana joka päivä pidetään Scrum-kokous, jossa käydään läpi, mitä kehittäjät ovat tehneet edellisenä päivänä, mitä he aikovat tehdä tänään ja mahdolliset ongelmat. Iteraatioiden jälkeen uudet ominaisuudet esitellään projektin omistajille. Kuvio 4 havainnollistaa Scrumin prosessia. [5.]



Kuvio 4. Scrum-projektinhallintamentelmän prosessi [5].

Scrum-malli korostaa asiakkaan osallistumista ja vaikutusta kehitysprosessin aikana. Asiakas näkee jokaisen iteraation jälkeen toimivan version tuotteesta. Tällöin asiakas voi muuttaa projektin vaatimuksia tai lisätä niitä kesken projektin. Scrum-projekti on myös asiakkaalle taloudellisesti turvallinen, koska asiakas voi rahoittaa vain yhden iteraation kerrallaan ja ongelmien syntyessä pitää senhetkisen version tuotteesta ja vaihtaa ohjelmistotoimittajaa. Perinteisissä projekteissa asiakas yleensä ostaa tuotteen kokonaisena eikä toimittajan vaihtaminen ole silloin helppoa. [5.]

Työlistan laatiminen

Työlista (product backlog) sisältää kaikki sovellukselle asetetut vaatimukset jaoteltuina selkeisiin komponentteihin. Komponenteille on määritelty työmääräarvio sekä liikearvo jonka mukaan ne priorisoidaan. Työlista saattaa myös sisältää tehtäviä, jotka eivät ole sovelluksen konkreettisia komponentteja, mutta liittyvät sovelluksen kehittämiseen. Esimerkki tämänlaisesta tehtävästä voisi olla palvelimen konfigurointi. Tuotteen omistaja on vastuussa työlistan laatimisesta. [10.]

Pyrähdykset

Scrum-projektissa töitä tehdään pyrähdyksissä (sprint), eli työt jaetaan määrätyn mittaisiin jaksoihin. Jaksojen pituus vaihtelee noin kahdesta neljään viikkoon. Neljä viikkoa on optimaalisin pyrähdyksen kesto, koska siinä ajassa on mahdollista tuottaa sen verran valmista ohjelmistoa, että sitä on mielekästä esitellä asiakkaalle, mutta työn määrä pysyy kuitenkin sen verran pienenä, ettei pyrähdyksen tehtäviä varten tarvitse kirjoittaa dokumentaatiota. Pyrähdykset ovat tehokas väline projektin aikataulutukseen ja niiden käyttö mahdollistaa resurssien hyödyntämisen tärkeimpiin komponentteihin. Jokaisen pyrähdyksen tavoitteena on tuottaa toimiva versio ohjelmistosta. Projektin jakaminen pyrähdyksiin tuo projektiin useamman aikarajan, jolloin aikataulussa pysyminen on helpompaa. [10.]

Jokaista pyrähdystä varten tehdään oma työlista (sprint backlog), johon valitaan koko projektin työlistasta pyrähdyksessä toteutettavat komponentit. Pyrähdykseen valitaan joukko työlistassa korkeimmalle priorisoituja komponentteja, jotka kehitysryhmä uskoo

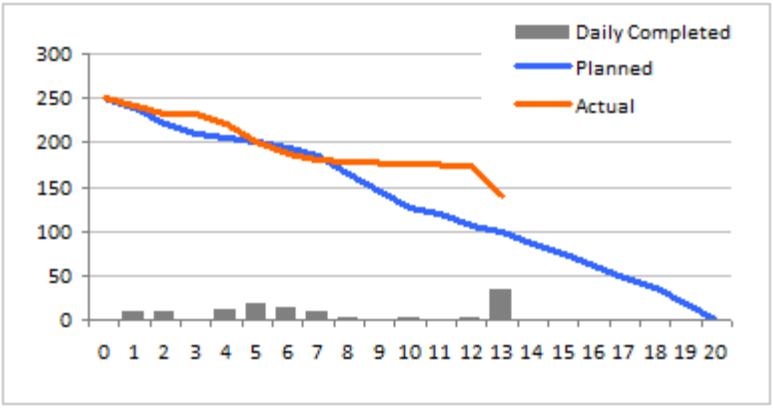
saavansa valmiiksi pyrähdyksen loppuun mennessä. Pyrähdyksen työlistaa varten komponentit jaetaan pienempiin tehtäviin. Tehtävien koon pitäisi olla työtunneissa mitattuna neljän ja 16 tunnin välillä. Pyrähdyksen työlista tulisi laatia niin, että siihen valitut tehtävät tuottavat toimivan version tuotteesta pyrähdyksen päätyttyä. Pyrähdyksen aikana pyrähdyksen työlistaan ei saa lisätä tehtäviä sen ulkopuolelta, uusia tehtäviä saa luoda ainoastaan listaan pyrähdyksen alussa lisättyjen komponenttien pohjalta. Pyrähdyksen lopussa kehitysryhmä esittelee sovellusta projektiryhmälle, jolloin projektin omistajat voivat arvioida tuloksia ja tarvittaessa tehdä muutoksia määrittelyyn ja työlistaan. [5; 10.]

Roolit Scrumissa

Scrum määrittelee kolme roolia: Scrum-mestari, Scrum-ryhmän jäsen ja tuotteen omistaja. Scrum-ryhmä tarkoittaa kehittäjäryhmää, joka koostuu ohjelmoijista. Scrum-projektissa työskennellään 5–9 hengen ryhmissä. Scrum-mestarin tehtävänä on palvella ryhmän jäseniä. Hän varmistaa, että ryhmä noudattaa Scrumin sääntöjä ja että tapaamiset kulkevat sulavasti. Scrum-mestarin tehtävänä on myös eliminoida kaikki työskentelyä hidastavat esteet. Jos ryhmän jäsen kohtaa ongelman, joka haittaa hänen työnsä edistymistä, on Scrum-mestari vastuussa ongelman ratkaisemisesta, jotta työt voivat jatkua esteettä. Scrum-mestari voi myös tehdä mestarin roolin ohella kehitystyötä eli ohjelmoida. Tuotteen omistaja hallitsee osakkaiden tuotteelle määrittelemiä vaatimuksia, priorisoi ne ja päättää mitkä ovat vaatimusten täyttämiseen hyväksyttävät kriteerit. Hän on myös vastuussa valmiin tuotteen toimivuudesta. Scrumia voidaan soveltaa niin, että Scrum-mestari ja tuotteen omistaja ovat sama henkilö, kuten Katsomon uudistusprojektissa tehtiin. [10.]

Dokumentointi

Scrumin tavoitteena on minimoida tuotettavan dokumentaation määrä. Scrum-projektissa on kuitenkin mahdollista tuottaa erilaisia dokumentteja, jotka kuvaavat projektin edistymistä. Burndown chart on taulukko, joka kuvaa tehtävien valmistumista suhteessa projektin etenemiseen. Kuviossa 5 on esimerkki burndown chartista. Jokaisen pyrähdyksen jälkeen tehdään raportti, joka kuvaa pyrähdyksen ja koko projektin edistymistä. [10.]



Kuvio 5. Esimerkki burndown -kaaviosta [12].

Kokoukset

Scrum-projektissa pidetään päivittäin nopea noin 15–30 minuuttia kestävä kokous. Kokous pidetään yleensä päivän alussa ja se voi olla hyvinkin epämuodollinen, esimerkiksi taukotilassa seisaalteen pidettävä tapaaminen. Tapaamisessa tiimin jäsenet kertovat Scrum-mestarille, mitä he ovat tehneet edellisen päivän aikana ja mitä he aikovat tehdä tulevan päivän aikana. Jos työskentelylle on jotain esteitä, ne kerrotaan tapaamisessa Scrum-mestarille ja Scrum-mestarin tulee ottaa nämä ongelmat ratkaistavakseen. Tämä varmistaa sen, että kehittäjät pystyvät keskittymään täysillä heidän työhönsä eli ohjelmointiin. [10.]

Myös ennen seuraavaa pyrähdystä pidetään kokous jossa päätetään pyrähdykseen tulevat tehtävät (sprint planning). Tehtävät valitaan tuotteen tehtävälistasta, josta ne siirretään erilliseen pyrähdyksen tehtävälistaan. Myös työmääräarviot käydään läpi ja sovitetaan kehitystiimin resursseihin. [10.]

Muita mahdollisia kokouksia ovat pyrähdyksen arviointi (sprint review), jossa pyrähdyksen tulokset esitellään asiakkaalle ja pyrähdyksen katselmointi (sprint retrospective), jossa kehitystiimin kesken arvioidaan pyrähdyksen saavutuksia ja mietitään, mitä voitaisiin parantaa seuraavassa pyrähdyksessä. [10.]

3.5 Projektinhallinta Katsomon uudistusprojektissa

3.5.1 Scrumin soveltaminen Katsomo-projektissa

Tässä projektissa sovellettiin Scrumia löyhästi. Scrumista poiketen projektissa oli projektipäällikkö, joka samalla hoiti myös Scrum-mestarin tehtävät ja toimi myös osittain tuotteen omistajana. Vaikka Scrumin ohjeiden mukaan projektipäällikköä ei kuuluisi olla, niin projektipäällikön käyttö tässä projektissa ei tuntunut vievän mitään Scrumin hyötyjä pois. Hajautetussa projektissa projektipäällikön puuttuminen olisi varmasti aiheuttanut ongelmia, koska usean yrityksen välisen yhteistyön onnistuminen vaati projektipäällikön auktoriteettia. [11.]

Kokousten suhteen Scrumia noudatettiin melko tiukasti. Joka päivä pidettiin puolen tunnin mittainen päivittäinen kokous, jossa jokainen kehittäjistä kertoi, mitä oli tehnyt edellisenä päivänä ja mitä hän aikoo tehdä tänään. Kokouksessa käytiin myös läpi kaikki työn etenemistä haittaavat ongelmat, jotka projektipäällikkö otti ratkaistavaksi. Yleensä suurin osa kokouksen ajasta menikin ongelmista puhumiseen. Projektissa pidettiin yhdistetty pyrähdyksen suunnittelu sekä arviointi. Ennen seuraavan pyrähdyksen alkua pidettiin kokous, jossa käytiin läpi edellisen pyrähdyksen tulokset ja laadittiin seuraavan pyrähdyksen työlista.

Pyrähdysten pituus vaihteli projektin eri vaiheiden intensiivisyyden mukaan. Projektin ensimmäisen vaiheen aikana käytettiin kahden viikon mittaisia pyrähdyksiä. Ensimmäisen vaiheen lopussa ennen ensimmäisen version julkaisua käytettiin viikon mittaisia pyrähdyksiä, koska projekti oli intensiivisessä vaiheessa ja työtahti oli tiukka. Myöhemmissä vaiheissa käytettiin kolmen viikon mittaisia pyrähdyksiä. Pyrähdykset erosivat normaaleista Scrum-pyrähdyksistä siten, että jokaisen pyrähdyksen tuotoksena ei ollut tarkoitus saada julkaisukelpoista ohjelmistoa, vaan jokaisen kolmen isomman vaiheen jälkeen tuotteen täytyi olla julkaisukelpoinen. Ensimmäinen vaihe koostui seitsemästä kahden viikon pyrähdyksestä. Ensimmäinen vaihe sisälsi suurimman osan sivuston toiminnallisuudesta ja se oli kaikista intensiivisin vaihe kehittäjien osalta. Ensimmäisen vaiheen jälkeen pyrähdysten pituutta kasvatettiin kolmeen viikkoon, koska projekti ei ollut enää niin intensiivisessä vaiheessa.

Projektissa työskenneltiin yhdessä viiden hengen ryhmässä, josta neljä henkilöä oli kehittäjiä ja yksi hoiti Scrum-mestarin ja projektipäällikön tehtävät. Tämän lisäksi norjalaiset osallistuivat projektiin noin viiden hengen ryhmässä.

3.5.2 Projektin aikatauluttaminen

Projekti jaettiin kolmeen noin kolmen kuukauden mittaiseen vaiheeseen. Jokaisen vaiheen lopputuloksena oli julkaisukelpoinen palvelu joka vietiin tuotantoon. Ensimmäiseen vaiheeseen valittiin mukaan palvelun toiminnan kannalta kriittisimmät komponentit. Näihin kuuluvat videoiden toisto, uudistettu ostoprosessi, omien käyttäjätietojen hallinta, ohjelmien omat sivut, kategoriasivut, hakutoiminnallisuus, videolistaukset ja ohjelmalistaukset. Toinen vaihe sisälsi käyttäjän henkilökohtaiset soittolistat, suosikkiohjelmien hallinnan ja erilaisia suositusominaisuuksia. Kolmas vaihe tulee sisältämään reaaliaikaisen videon aikajanalla tapahtuvan kommentointiominaisuuden.

Projektin aikataulu ei toteutunut suunnitellusti. Ensimmäisen vaiheen julkaisu myöhästyi noin kuukaudella aikataulusta ja toisen vaiheen julkaisua tarvitsi siirtää vain viikolla. Aikataulun kanssa tuntui olevan jatkuvasti pieniä ongelmia jotka johtuivat enimmäkseen ongelmista töiden synkronoinnissa Cybercomin ja Norjan välillä. Näitä käsitellään tarkemmin luvussa 3.5.5.

3.5.3 Katsomo-projektin työlistan laatiminen

Katsomo-projektin alussa sovellus jaettiin itsenäisiin komponentteihin konseptisuunnitelman ja rautalankamallien perusteella. Kehitystiimin jäsenet saivat valita haluamansa komponentit omaan vastuualueeseensa. Tämän jälkeen komponentit jaettiin pienempiin tehtäviin jotka lisättiin JIRA-tehtävänhallintajärjestelmään ja niille annettiin alustava työmääräarvio. Työmääräarvion perusteella tehtävät pystyttiin jakamaan tasaisesti pyrähdyksien kesken. Taulukossa 1 näkyy lista kaikista komponenteista. Projekti ei ole vielä päättynyt, ja tätä kirjoittaessa projektille oli tehty 467 tehtävää 26:lle komponentille.

Taulukko 1. Projektin jako komponentteihin.

Komponentti	Kuvaus
All programs layer	ohjelmavalikko
Backend and api	taustajärjestelmään tehtävät muutokset
Category pages	kategorioiden pääsivut
Error page	kustomoidut virhesivut
Front page	palvelun etusivu
Help & support	ohje- ja palautesivut
Legacy products	vanhat tuotteet jotka halutaan säilyttää muuttumattomina
Main carousel	sisältöjen nostamiseen käytettävä kuvakaruselli
Measurements & statistics	käyttäjämäärien ja sivulatausten mittaamista
Multinews integration	sisällönhallintajärjestelmän integrointi
Navigation, header & footer	sivuston navigaatio ja muut kiinteät elementit
New feature ideas	mahdolliset kehittäjien huomaamat parannusideat
Notifications	käyttäjille näytettävät virheilmoitukset
Playback queue	käyttäjän henkilökohtainen ohjelmaputki
Program page	yksittäisten ohjelmien omat sivut
Registration & purchase flow	käyttäjien rekisteröinti sekä tuotteiden ostoprosessi
Search	sivuston hakutoiminto
Sharing	videoiden facebook-, twitter- ja sähköpostijakaminen
Silverlight player	videosoittimeen liittyvät muutokset
Testing	sivuston testaus
Theme pages	kustomoitavat teemasivut
Timeline commenting	videon aikajanalla tapahtuva koimmentointi
UltraVenetsia integration	MTV3:n ohjelmatietokannan integrointi ohjelmatietoja hakemista varten
User profile	käyttäjän profiilisivut
Video grid	komponentti videolistausten näyttämistä varten
Videoplaza integration	mainostenhallintajärjestelmän integrointi

Esimerkkinä tehtävien jakamisesta voidaan käyttää Katsomon videotaulukkokomponenttia. Tämä komponentti jaettiin taulukossa 2 näkyvään 14 tehtävään. Projektin ede-

tessä tehtäviä täytyi luoda lisää ja jotkin aiemmin luoduista tehtävistä todettiin turhiksi eikä niitä toteutettu.

Taulukko 2. Videotaulukkokomponentin jako tehtäviin.

Tehtävä	Tehtävän nimi
MTVKATSOMODEV-127	Pagination in listings
MTVKATSOMODEV-122	Update HTML produced by default.mid.listContent –tile
MTVKATSOMODEV-28	HTML/CSS layout
MTVKATSOMODEV-50	Info popup HTML/CSS/JS
MTVKATSOMODEV-73	Fetch the data used in the info popup
MTVKATSOMODEV-68	Tech spec for filtered video grid views
MTVKATSOMODEV-116	Include age limit in the info popup
MTVKATSOMODEV-66	Script and UltraVenetsia modules for retrieving JSON data to match all filtered views
MTVKATSOMODEV-43	Video grid pagination, expanding and scalability
MTVKATSOMODEV-52	Javascript functionality
MTVKATSOMODEV-31	Configure HTML using tiles
MTVKATSOMODEV-69	Find out how the videos about to expire filter can be implemented - where's the data?
MTVKATSOMODEV-98	Make video grid adjust to context. Different number of items or rows, different dimensions
MTVKATSOMODEV-129	Custom HTML-templates in AJAX-requests

3.5.4 Ketterän kehityksen tuomat edut Katsomo-projektissa

Isoin etu, jonka Scrumin käyttö toi projektille, oli mahdollisuus tarkastella projektin edistymistä kahden viikon välein pyrähdysten välillä ja esitellä tuloksia johtoryhmälle. Tarkasteluissa huomattiin myös ongelmat nopeasti ennen kuin ne ehtivät eskaloitua isommiksi ja ongelmiin ehdittiin puuttua ajoissa. Käytännössä tämä tarkoittaa sitä, että kun huomasimme jonkin komponentin tuottavan niin paljon ongelmia, että sitä ei kan-

nata toteuttaa, se priorisoitiin uudelleen tai pudotettiin kokonaan pois työlistalta, jolloin resursseja ei mennyt hukkaan ongelman selvittelyyn. [11.]

Ketterä kehitys mahdollisti myös sen, että projektin toteutusvaihe pystyttiin aloittamaan ilman teknistä määrittelyä pelkän toiminnallisen määrittelyn pohjalta. Tekninen määrittely tehtiin vasta komponenttien ohjelmoinnin yhteydessä, jolloin tehtiin ainoastaan tarpeellinen määrittely. Koska teknistä määrittelyä tehtiin projektin edetessä vaiheittain, minimoitiin mahdolliset virheet määrittelyssä.

Vaikka projektin aikataulu ei pitänyt suunnitellusti, Scrumin käyttö kuitenkin helpotti aikataulussa pysymistä, koska töitä tehtiin iteroiden ja tuloksia seurattiin jatkuvasti.

3.5.5 Ongelmat Katsomo-projektin projektinhallinnassa

Yksi isoimmista ongelmista, joka Scrumin käytössä tässä projektissa ilmeni, oli Scrumin käytön epäsynkronisuus Cybercomin sekä norjalaisten välillä. Pyrähdykset olisi pitänyt saada mahdollisimman yhtenäisiksi yritysten välillä ja pyrähdyksien suunnittelut olisi pitänyt tehdä yhdessä. Nyt molemmat yritykset pitivät omat pyrähdyksensuunnittelukokouksensa ja tästä johtuen pyrähdykset eivät olleet keskenään yhtenäisiä. [11.]

Kun seuraavaa pyrähdystä suunniteltiin, otettiin siihen mukaan sen verran tehtäviä, että ne oli mahdollista toteuttaa pyrähdyksen aikana. Sen jälkeen norjalaisilta tilattiin näihin tehtäviin liittyvät taustajärjestelmän muutokset. Norjalaiset sitten toteuttivat pyrähdyksessään sen verran kuin ehtivät, ja jäljelle jäänet työt siirrettiin aina seuraavaan pyrähdykseen. Tästä johtuen jouduttiin usein odottamaan pitkään töiden etenemistä, koska kehitysryhmä Suomessa oli riippuvainen taustajärjestelmän muutoksista, jotka olivat myöhässä. Oikea tapa olisi ollut suunnitella pyrähdykset yhdessä yritysten kesken, jolloin pyrähdykseen olisi otettu vain sellaisia toisistaan riippuvia töitä, jotka molemmat yritykset ehtivät varmasti toteuttamaan pyrähdyksen aikana. [11.]

Norjasta tilatut muutokset toimitettiin pakettina, joka asennettiin ja testattiin kehitysympäristössä. Pakettien mukana ei yleensä toimitettu selkeätä listaa tehdyistä muutoksista, mikä hankaloitti muutosten integrointia ja testausta. [2.]

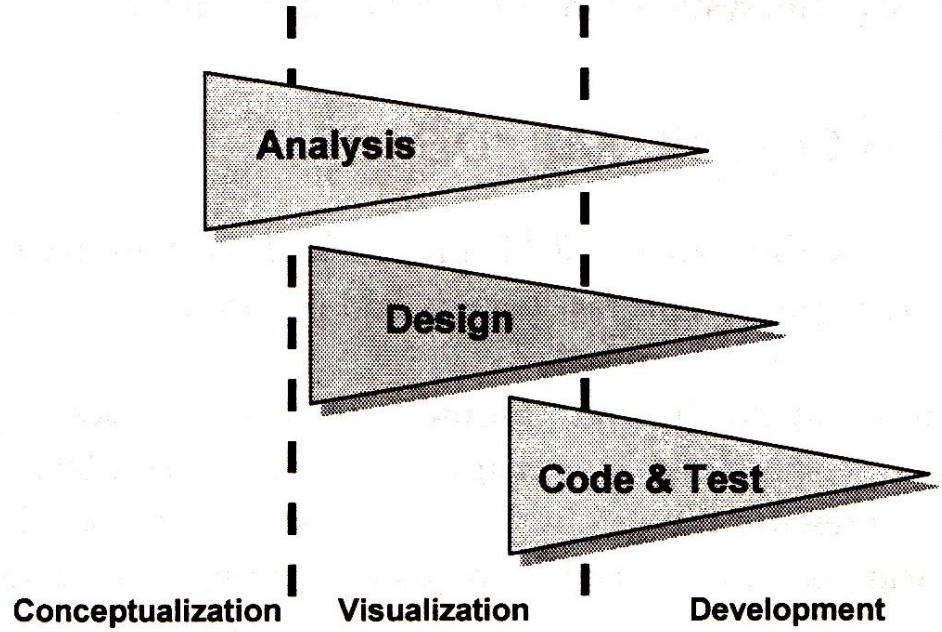
Ongelmaksi muodostui myös se, että taustajärjestelmästä vastaavia norjalaisia ei konsultoitu konseptointivaiheessa. Tästä johtuen konseptissa oli paljon ominaisuuksia jotka olivat ristiriidassa taustajärjestelmän kanssa tai erittäin hankalia toteuttaa. Konseptia tehneen muotoilutoimiston olisi ollut hyvä pystyä keskustelemaan norjalaisten kanssa ja kysyä pystyykö heidän suunnittelemia ominaisuuksia toteuttamaan helposti. Tällöin toteutuskelvottomia ominaisuuksia olisi jätetty pois jo konseptointivaiheessa ja niiden tilalle olisi voitu suunnitella uusia toteutuskelpoisia ominaisuuksia. Nyt toteutuskelvottomat ominaisuudet huomattiin vasta toteutusvaiheessa ja tästä syystä lopullinen sovellus oli joiltain osin konseptin näkökulmasta hieman keskeneräinen. [11.]

Projektissa noudatettiin konseptia ja toiminnallista määrittelyä erittäin tarkasti eikä muutoksia sallittu, elleivät ne olleet teknisesti mahdottomia toteuttaa. Tämän takia Scrumin tuomaa helppoa muutosten hallintaa ei pystytty hyödyntämään projektissa kovinkaan hyvin, koska muutoksia ei juurikaan haluttu tehdä. Scrumin hyödyt olisivat tulleet vielä paremmin esiin projektissa, jossa ei olisi ollut näin kattavaa konseptia ja toiminnallista määrittelyä alussa, vaan sivuston toiminnallisuus olisi muovautunut projektin edetessä asiakkaan vuorovaikutuksesta.

4 Rapid Application Prototyping -kehitysmalli

4.1 Rapid Application Prototyping (RAP)

Rapid Application Prototyping (RAP) on kehitysprosessi jonka tarkoituksena on saada mahdollisimman nopeasti aikaan fyysinen tai visuaalinen versio tuotteesta. RAP-prosessi koostuu kolmesta vaiheesta jotka ovat konseptointi, visualisointi ja toteuttaminen. Ensimmäinen vaihe on konseptointi, jossa määritellään järjestelmän vaatimukset ja tehdään konseptisuunnitelma. Toinen vaihe on visualisointi, jossa konseptisuunnitelman pohjalta tehdään visuaalinen suunnitelma. Tämä sisältää sovelluksen rautalankamallit ja käyttöliittymän toiminnallisien kuvauksen. Rautalankamallien perusteella tehdään prototyyppi jonka toiminnallisuuksia aletaan ohjelmoimaan kolmannessa vaiheessa eli toteutuksessa. Kuviossa 6 näkyvät kaikki työvaiheet aikajanalle aseteltuna. Kuten kuvasta näkyy, vaiheet voivat mennä päällekkäin. Kolmioiden pinta-alat kuvaavat vaiheen työmäärää, joka jokaisen vaiheen kohdalla vähenee projektin edetessä. [13.]



Kuvio 6. Rapid Application Prototyping prosessina [13].

RAP-projekti tuottaa visuaalisen version tuotteesta 2–4 kertaa nopeammin kuin perinteiset kehitysprosessit, joissa käyttöliittymä toteutetaan vasta kun taustajärjestelmä on valmis. Asiakas ja käyttäjät pääsevät kokeilemaan prototyyppiä jo paljon ennen kuin kaikki toiminnallisuudet ovat valmiina. Tämän ansiosta tuotetta voidaan testauttaa asiakkaalla huomattavasti aikaisemmin, jolloin mahdolliset muutokset eivät aiheuta niin suurta työmäärää taustajärjestelmän muuttamiseen. [13.]

RAP-mallia voidaan soveltaa kahdella tavalla: Throw-away prototyping ja Evolutionary Prototyping. Throw-away prototyping -mallissa prototyyppejä käytetään ainoastaan sovelluksen suunnittelun ja esittelyn apuna. Prototyyppejä arvioimalla saadaan muodostettua lista sovellukseen tulevista ominaisuuksista ja komponenteista. Sen jälkeen prototyypit hylätään ja lopullinen sovellus rakennetaan alusta alkaen. Usein prototyypit tehdään eri ohjelmointikielillä ja tekniikoilla kuin lopullinen sovellus. [14]

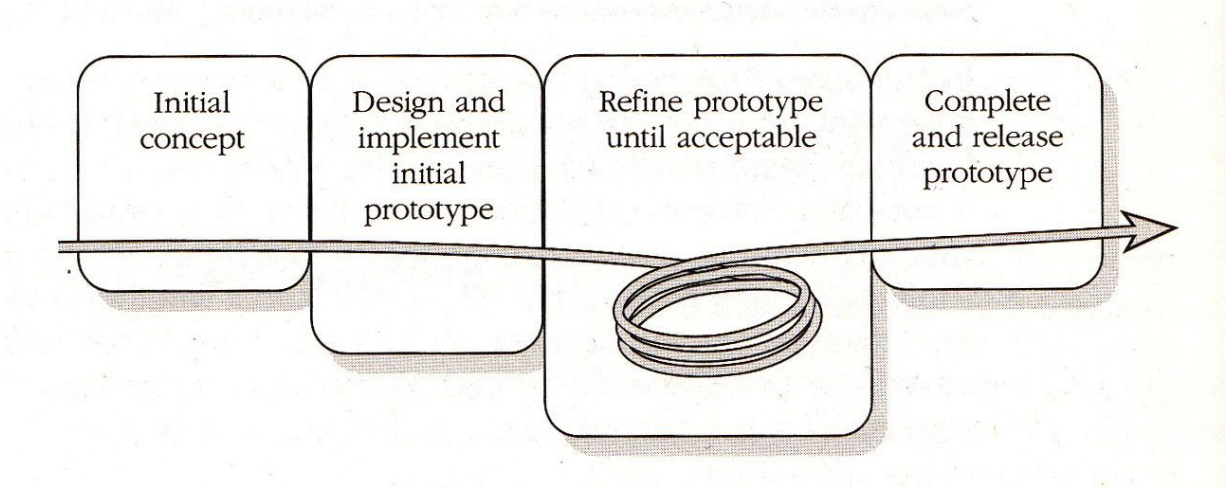
Evolutionary Prototyping -mallissa prototyyppejä arvioidaan, kehitetään ja niistä jalostetaan lopullinen sovellus. Prototyyppejä kehitetään iteraatioissa jolloin lopullinen so-

vellus valmistuu inkrementaalisesti kasvaen. Prototyyppi hylätään vain, jos kyseistä komponenttia ei toteuteta lopulliseen sovellukseen. [14.]

Usein RAP-termillä viitataan Evolutionary Prototyping -malliin, mutta RAP on vain yleinen kuvaus, joka viittaa kaikkiin prototyyppejä hyödyntäviin kehitysmenetelmiin. Evolutionary Prototyping on näistä kuitenkin yleisin. Evolutionary Prototyping –mallia kuvataan tarkemmin seuraavassa luvussa. [13.]

4.2 Evolutionary Prototyping

Evolutionary Prototyping on yksi RAP-prosessin käytännöistä. Evolutionary Prototyping -prosessin periaatteena on rakentaa ensin tietyt valitut sovelluksen osat ja sen jälkeen loput osat niiden päälle. Usein ensimmäisenä kannattaa rakentaa sovelluksen näkyvimmät ja riskialttiimmat osat. Yleensä käyttöliittymä on sovelluksen näkyvin ja riskialttiin osa, joten Evolutionary Prototyping -prosessissa tyypillistä on rakentaa käyttöliittymä ensin ja kehittää asteittain toiminnallisuuksia sen pohjalta. Prototyypin rakentaminen tapahtuu asteittain, jolloin asiakas pääsee seuraamaan prototyypin kehitystä ja tarvittaessa pyytämään muutoksia. Kuvio 7 kuvaa Evolutionary Prototyping -prosessia. [6.]



Kuvio 7. Evolutionary Prototyping prosessina [6].

Evolutionary Prototyping on on melko riskitön malli, mutta se tuo kuitenkin muutamia riskejä projektiin. Koska asiakas näkee käyttöliittymän prototyypin kehittyvän nopeasti, saattaa hänelle tulla epärealistisia odotuksia koko tuotteen valmistumisen aikataulusta.

Usein käyttöliittymän rakentaminen on kuitenkin huomattavasti nopeampi prosessi kuin taustajärjestelmän ohjelmointi eikä asiakas välttämättä ymmärrä sitä. [6]

Prototyyppi voi myös aiheuttaa epärealistisia odotuksia suorituskyvyn suhteen. Prototyypin ei tarvitse tehdä kaikkea työtä eikä käsitellä kaikkea tietoa, mitä valmiin tuotteen täytyy. Koska prototyyppi tuntuu toimivan suorituskyvyllisesti hyvin, saattaa kehittäjä unohtaa kiinnittää suorituskykyyn huomiota ja suorituskykyongelmat tulevat esiin vasta lopullisen tuotteen valmistuessa. [6.]

Prototyyppikehitys saattaa johtaa myös huonoon arkkitehtuuriin ja heikkoon ylläpidettävyyteen. Isoin syy tähän on hyvin nopea kehitys ja ”koodaa ja korjaa” -ajattelutapa, jotka usein ovat läsnä prototyyppikehityksessä. Ketterissä kehitysmenetelmissä tekniseen määrittelyyn käytetään yleensä vähemmän aikaa ja määrittelyä tehdään projektin edetessä, joten tämä voi johtaa sekavaan koodiin ja huonoon ylläpidettävyyteen. Näiltä riskeiltä vältytään, jos kehittäjät ovat kokeneita ja hyödyntävät parhaita ratkaisumalleja. [6.]

4.3 Prototyyppien käyttö Katsomo-projektissa

Katsomon uudistusprojektissa ei käytetty tietoisesti mitään kehitysmallia, mutta siinä toimittiin melko pitkälti RAP- ja Evolutionary Prototyping -mallien mukaan. Mallien mukainen työtapa valittiin osittain siksi, että taustajärjestelmän kehitys tapahtui ulkoistettuna ja taustajärjestelmän muutokset olivat helpompia toteuttaa, kun komponenttien käyttöliittymistä oli olemassa jo prototyypit. Osa käyttöliittymän HTML-koodista täytyi upottaa taustajärjestelmän koodiin, joten taustajärjestelmän muutoksia määriteltäessä käyttöliittymän HTML-koodi tuli olla jo valmiina. Prototyyppien käyttö on muutenkin erittäin yleistä web-sovelluskehityksessä.

4.4 Videotaulukkokomponentin prototyyppi

Konseptointi

Videotaulukko on komponentti, jolla esitetään videolistauksia sivuilla. Taustajärjestelmä palauttaa sille annettujen parametrien perusteella listan videoita, jotka näytetään käyttäjälle videotaulukossa. Videoita voidaan järjestää ja suodattaa käyttöliittymässä.

Videotaulukon rakentaminen mukaili Evolutionary Prototyping –prosessia. Ensimmäinen prototyyppi sisälsi videotaulukon toiminnallisen kuvauksen ja liitteessä 1 näytetyn käyttöliittymän rautalankamallin. Toiminnallinen kuvaus sisälsi kuvaukset kaikista komponentin näkymistä ja käyttöliittymän toiminnoista. Tämän jälkeen prototyyppiä kehitettiin tekemällä liitteessä 2 näkyvän käyttöliittymän graafinen ulkoasu rautalankamallin pohjalta. Tekniseen toteutukseen ei otettu tässä vaiheessa vielä kantaa. Konsepti ja käyttöliittymän graafinen ilme tilattiin muotoilutoimistolta.

Teknisen prototyypin rakentaminen

Käyttöliittymän graafisen mallin pohjalta ryhdyttiin rakentamaan käyttöliittymän HTML-rakennetta ja CSS-tyylejä. Kun HTML-rakenne oli valmis, joitakin käyttöliittymän JavaScript-toiminnallisuuksia toteutettiin, jotta prototyyppi havainnollistaisi paremmin komponentin toimintaa. Prototyyppiä varten luotiin staattista testidataa, jotta sen toimintoja pystyttiin testaamaan.

Kun prototyyppi oli valmis HTML-, CSS- ja JavaScript-koodin osalta, tehtiin komponentin toiminnan tekninen määrittely. Tekninen määrittely sisälsi tarkan kuvauksen komponentin toiminnasta ja taustajärjestelmään kohdistuvista vaatimuksista. Tähän kuului lista komponentille annettavista parametreista, kuvaukset listauksista joita komponentin kuuluu tuottaa ja AJAX-toiminnallisuuden määrittely. Määrittely sisälsi ainoastaan komponentille asetettavat vaatimukset eikä se ottanut kantaa tekniseen toteutustapaan. Norjalaiset saivat itse suunnitella toteutuksen komponentin määrittelyn ja prototyypin perusteella. Tekninen määrittely ja käyttöliittymän prototyyppi toimitettiin Norjaan taustajärjestelmästä vastaavalle yhtiölle, jossa määrittelyn ja prototyypin ohjeistuksella tehtiin tarvittavat muutokset taustajärjestelmään.

Alun perin komponentin koodin ajateltiin sijaitsevan kokonaan taustajärjestelmässä, mutta norjalaiset ehdottivatkin että osa ylimmästä logiikasta tehtäisiinkin meidän toimesta Velocity-sivupohjassa. Kun taustajärjestelmän muutokset olivat valmiit, tehtiin videotaulukkoa varten Velocity-sivupohjan osa, johon lisättiin prototyypin HTML-koodi ja Velocity-syntaksin avulla tulostettiin taustajärjestelmän palauttamaa dataa HTML-koodin sekaan. Prototyypin käyttö mahdollisti lopullisen toteutustavan valinnan vasta erittäin myöhäisessä vaiheessa ilman erityistä vaivaa tai ongelmia. Prototyyppien esittely norjalaisille auttoi heitä suunnittelemaan muutosten teknistä toteuttamista, koska käyttöliittymän prototyyppi havainnollistaa komponentin toimintaa paremmin kuin pelkkä tekninen määrittely.

Prototyyppikehityksen tuomat hyödyt

Sen lisäksi että prototyyppien käyttö helpotti norjalaisten kanssa työskentelyä, se toi myös muita hyötyjä. Koska prototyyppiä rakennettiin pienissä paloissa ja asiakas pystyi testaamaan prototyyppiä jatkuvasti, pystyttiin komponentin toteuttamiseen vaikuttaa lennosta. Koska konseptissa oli määritelty joitain ominaisuuksia, jotka olivat mahdottomia toteuttaa taustajärjestelmän tai tarvittavan datan puuttumisen vuoksi, pystyttiin nämä ominaisuudet tiputtamaan helposti pois.

5 Tehtävänhallinta

5.1 Tehtävänhallintajärjestelmien perusteet

Tehtävänhallintajärjestelmä on ohjelmisto, joka pitää yllä listaa tehtävistä joita työntekijät tekevät. Tehtävänhallintajärjestelmiä käytetään usein vikapäivystyksen tai puhelintuen työkaluna, johon asiakkaan ilmoittamat ongelmat raportoidaan ja välitetään eteenpäin oikeille henkilöille. Usein tehtävänhallintajärjestelmiä käytetään web-pohjaisen käyttöliittymän avulla, ja niitä on tarjolla valmiiksi ylläpidettyinä palveluntarjoajan palvelimilla tai omalle palvelimelle asennettavana ohjelmistona. Tunnettuja tehtävänhallintajärjestelmiä ovat Atlassianin kehittämät JIRA ja Confluence sekä yksinkertaisempi 37signalsin kehittämä Basecamp. Näistä JIRA soveltuu hyvin Scrum-projektin

tehtävänhallinnan alustaksi sen monipuolisten ominaisuuksien sekä Greenhopper-lisäosan ansiosta. [15.]

5.2 JIRA-tehtävänhallintajärjestelmä

Ominaisuudet

JIRA on tehtävänhallintajärjestelmä, johon projekti voidaan kirjata pienempinä komponentteina, jotka on taas jaettu pienempiin tehtäviin eli tiketteihin. Jokainen tiketti sisältää tehtävän kuvauksen, ja tikettiin pystyy kirjoittamaan kommentteja. Tiketille määritellään toteuttaja, tiedottaja ja katsojia. Tiedottaja on yleensä henkilö joka on syöttänyt tehtävän JIRAan, ja hän yleensä valvoo tehtävän edistymistä. Katsojat ovat sivullisia, jotka ovat jollain tavoin kiinnostuneita tehtävästä. Kaikki osapuolet saavat sähköpostilla ilmoituksen, kun tehtävää muokataan tai kommentoidaan. [16.]

Työnkulku

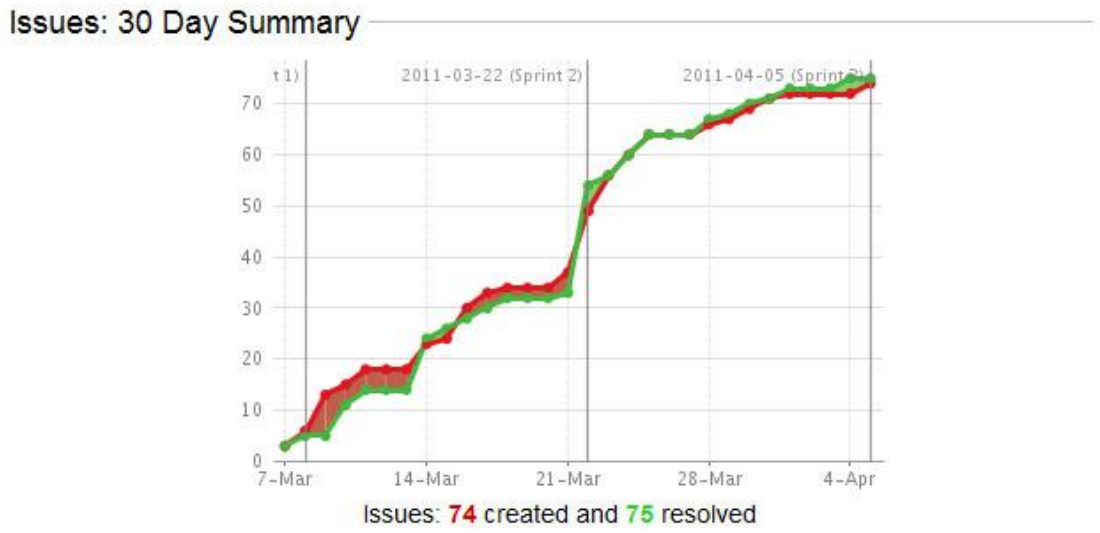
Kun JIRAan syötetään tiketti, sille määritellään tehtävän kuvaus ja henkilö, joka tehtävän toteuttaa. Tikettiin määritellään myös, mihin komponenttiin se liittyy ja minkä pyrähdyksen aikana tehtävän on tarkoitus valmistua. Tiketille voidaan määritellä myös arvioitu työmäärä tunneissa. Vasta luodun tiketin tila on avoin (open). Kun tiketin toteuttaja aloittaa tehtävän tekemisen, hän muuttaa tiketin tilan käynnissä olevaksi (in progress). Kun tehtävä on valmis, se merkitään valmistuneeksi (resolved). Kun tiedottaja on todennut tehtävän täyttävän sille asetetut vaatimukset, hän merkitsee tiketin suljetuksi (closed). Jos tehtävän toteutuksessa on puutteita, tiedottaja voi uudelleen avata tiketin, jolloin sen tila on uudelleenavattu (reopened). Tikettejä voidaan selata niiden tilan perusteella.

Tehtävän toteuttaja voi merkitä tehtävän toteuttamiseen käyttämänsä työtunnit tehtävän tuntikirjaukseen. Tällöin voidaan seurata komponenttiin tai koko projektiin kulutettuja työtunteja ja seurata työmääräarvioiden toteutumista. Työmääräarvioiden kirjaaminen tikettiin helpottaa myös pyrähdysten valmistelua, koska järjestelmä näyttää pyrähdykseen lisättyjen tikettien yhteenlasketun työmäärän.

5.3 JIRA-tehtävänhallintajärjestelmän käyttö Katsomo-projektissa

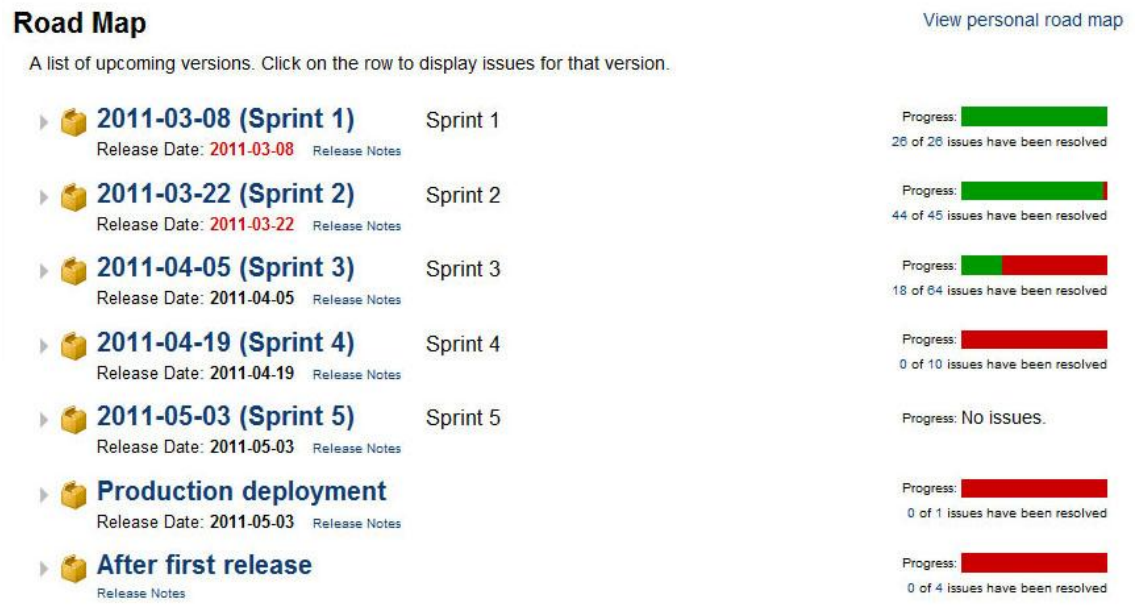
Projektin alussa sivuston komponentit kirjattiin JIRAan ja jokaiselle komponentille merkittiin niistä vastaava henkilö. Sen jälkeen työlista kirjattiin JIRAan, ja komponenttien alle luotiin niihin liittyvät tehtävät. Kun tehtävälista oli luotu, kehittäjät kävivät kirjaamassa työmääräarviot jokaiselle tehtävälle. Työmääräarvion avulla projektipäällikkö pystyi jakamaan tehtävät tasaisesti pyrähdyksiin. Kun kehittäjä oli saanut tehtävän valmiiksi, hän merkitsi käyttämänsä tunnit tehtävän tuntikirjaukseen. Tuntikirjauksista luovuttiin kuitenkin aika pian koska siitä saatavan informaation koettiin turhaksi.

JIRAn avulla pystyy tuottamaan erilaisia dokumentteja ja näkymiä, jotka kuvaavat projektin edistymistä. Kuviossa 8 näkyy 30 päivän ajalta, kuinka paljon tehtäviä on saatu valmiiksi ja kuinka paljon uusia tehtäviä on luotu.



Kuvio 8. Kuvankaappaus 30 päivän näkymästä JIRA-järjestelmässä

Kuviossa 9 on "road map" -näkymä, joka näyttää menneet ja tulevat pyrähdykset ja niiden tehtävien tilanteen. Kuvassa on meneillään kolmas pyrähdys, jonka tehtävistä on vajaa kolmas osa toteutettu. Kuvata näkyy myös, että jo päättyneessä toisessa pyrähdyksessä on yksi tehtävä, joka ei ole vielä valmistunut.



Kuvio 9. Roadmap-näkymä JIRA-järjestelmässä

Edellisessä kuvassa oleva näkymä toimii myös valmistuneiden tehtävien arkistona, josta niitä on tarvittaessa helppo kaivaa esiin. Tämä huomattiin tärkeäksi ominaisuudeksi, koska tehtäviin kasaantui usein pitkiä keskusteluita, joista oli hyötyä myöhemmin muita tehtäviä tehtäessä.

Tässä projektissa käytettiin myös useampaa Basecamp-tehtävänhallintajärjestelmää toisen ja kolmannen vaiheen aikana. Tämä johtui siitä, että suurin osa toisen ja kolmannen vaiheen toteutuksesta tapahtui Norjassa, ja siksi jouduttiin käyttämään heidän tehtävänhallintajärjestelmää. Myös Palmu Inc. -muotoilutomiston kanssa käytettiin heidän Basecamp-järjestelmäänsä, joten yhteensä projektissa oli käytössä kolme erillistä tehtävänhallintajärjestelmää. Tästä aiheutui ongelmia, koska usein tehtävät kirjattiin ensin JIRAan, josta ne siirrettiin jompaankumpaan Basecampiin. Kun sama tehtävä oli kahdessa paikassa, täytyi molempien tila muistaa päivittää ja usein tehtäviä jäi roikkumaan JIRAan, vaikka ne oli jo ratkaistu Basecampissa. Saman tehtävänhallintajärjestelmän käyttö olisi varmasti helpottanut tehtävänhallintaa.

5.4 Greenhopper-lisäosan käyttö Katsomo-projektissa

GreenHopper on Jiran lisäosa, joka mahdollistaa Scrum-projektin hallinnan JIRAssa. GreenHopper on kätevä työkalu tehtävälistan hallintaan, pyrähdysten suunnitteluun, projektin seurantaan ja dokumenttien tuottamiseen. [17.]

GreenHopper sisältää planning board -näkymän, jonka avulla tehtävälistaa ja pyrähdyksen tehtävälistoja voidaan hallita helposti. Tehtävät voi näyttää tekijän, komponentin tai version mukaan. Tehtäviä voi muokata ja niiden prioriteettia voi muuttaa helposti liikuttamalla niitä näkymässä. Myös uusien tehtävien lisääminen onnistuu Greenhopper-näkymässä helpommin kuin tavallisesti. [17.]

5.5 Basecamp-tehtävänhallintajärjestelmä

Basecamp on 37signals-yhtiön kehittämä ja vuonna 2004 julkaistu web-pohjainen tehtävänhallintajärjestelmä. Basecampissa työt jaetaan ”to-do” -tehtäviin joista voidaan muodostaa erilaisia listoja. Tehtävälle voi nimetä siitä vastaavan henkilön ja listan käyttäjiä jotka saavat sähköpostiin muistutuksia kun tehtävää päivitetään. Käyttäjät voivat kirjoittaa tehtävään kommentteja ja liittää siihen tiedostoja. Tehtävällä on kaksi tilaa, avoin ja suljettu. Kun tehtävä suljetaan, se häviää listalta. [18.]

Basecampista löytyy myös kalenterinäkymä jossa näkyy kaikki tehtävät niille määriteltyjen valmistumisaikojen mukaan. Basecamp sisältää myös writeboard-työkalun dokumenttien hallintaan. Siinä useat käyttäjät voivat muokata samaa dokumenttia sulavasti. Basecampin avulla voidaan myös jakaa tiedostoja käyttäjien kesken.

5.6 Basecampin ja JIRAn erot

JIRAan verrattuna Basecamp sisältää paljon vähemmän ominaisuuksia ja on huomattavasti yksinkertaisempi järjestelmä.

Basecampissa ei ole mahdollisuutta kirjata työtunteja tehtäville, joten työmäärien seuraaminen sen avulla on mahdotonta. Basecampin avulla ei pysty myöskään tulostamaan minkäänlaisia raportteja eikä kaavioita. Se ei myöskään tarjoa Scrum-

projektinhallintamenetelmää tukevia työkaluja, kuten JIRAn Greenhopper-lisäosa. Tehtäville ei voi myöskään määritellä mitään lisätietoja, kuten tehtävän tyyppiä, komponenttia tai avainsanoja. Tämän vuoksi tehtävien hakeminen järjestelmästä on vaikeaa. Ainoa tapa hakea tehtäviä Basecamp-projektista on vapaa tekstihaku. JIRAssa tehtäviä voi selata esimerkiksi tehtävän tekijän, komponentin, avainsanojen ja pyrähdyksen perusteella sekä käyttämällä SQL-tyyppisiä hakulausekkeita. Kun tehtävä oli merkitty valmistuneeksi, sitä oli todella hankala enää löytää Basecampista.

Basecampin halvin lisenssi maksaa 44 dollaria ja kallein 149 dollaria. Kumpikaan lisenssi ei sisällä käyttäjärajoituksia. JIRA:n halvin lisenssi maksaa 50 dollaria ja kallein 2000 dollaria. Halvin lisenssi on 11–15 käyttäjälle ja kallein 501–2000 käyttäjälle. Isommalle yritykselle Basecampin käyttö tulee huomattavasti halvemmaksi koska se ei sisällä käyttäjärajoituksia. Hinta ja yksinkertaisuus jäävätkin Basecampin ainoiksi eduiksi verrattuna JIRA:aan. [16; 18.]

JIRA on saatavana valmiiksi ylläpidettynä Atlassianin palvelimilla tai sen voi hankkia omalle palvelimelle asennettavana ohjelmistona jonka ylläpito tapahtuu itse. Basecamp on aina ylläpidettynä 37signalsin palvelimilla. Itsenäinen ylläpito omilla palvelimilla on hyvä vaihtoehto isolle yritykselle, joka käsittelee arkaluontoista tietoa tehtävänhallintajärjestelmässään. [16; 18.]

6 Yhteenveto

Insinöörityössä uudistettiin Katsomo-videopalvelu ja esiteltiin projektissa käytettyjä ketterän ohjelmistokehityksen menetelmiä. Tavoitteena oli saada palvelulle suunnitellut muutokset toteutettua sille asetetun aikataulun mukaisesti. Haastetta toi ohjelmakoodin kehityksen jakautuneisuus eri yritysten kesken ja tiukka aikataulu. Projekti onnistui lopputuloksen kannalta erittäin hyvin, joskin aikataulusta myöhästyttiin hieman. Projektinhallinta ja menetelmien käyttö oli myös tehokasta, mutta yritysten välisessä yhteistyössä ilmeni kuitenkin joitain ongelmia.

Tässä projektissa kävi selkeästi ilmi, että ketterät menetelmät, kuten Scrum ja prototyyppien käyttö, sopivat hyvin juuri web-sovelluskehitykseen, jossa aikataulut ovat

tiukkoja ja muutokset kesken kehityksen todennäköisiä. Projektissa huomattiin myös, että saadakseen hyötyä näiden mallien käytöstä, täytyy niitä noudattaa tarkasti ja samojen pelisääntöjen mukaan. Nyt suurimmat ongelmat muodostuivat kahden yrityksen epäsynkronisesta yhteistyöstä. Tätä olisi voitu parantaa käyttämällä samaa Scrum-prosessia molempien yritysten kesken.

Projektissa hyödynnettiin prototyyppejä kehityksessä, mikä on yleistä web-sovelluskehityksessä. Katsomon uudistusprojektissa Prototyyppien käyttö toi erityistä lisäarvoa helpottaen saman komponentin eri osien kehittämistä kahdessa yrityksessä. Prototyypin ja teknisen määrittelyn toimittaminen taustajärjestelmää kehittävälle yhtiölle helpotti heidän työtään, koska käyttöliittymän prototyyppi havainnollisti komponentin toimintaa paremmin kuin pelkkä tekninen määrittely.

Tehtävänhallintajärjestelmät olivat projektissa keskeisessä osassa kommunikoinnissa ja töiden dokumentoinnissa. Projektissa jouduttiin käyttämään kolmen eri yrityksen omia tehtävänhallintajärjestelmiä, mikä hankaloitti töiden seuraamista ja arkistointia. Tätä olisi helpottanut tehtävänhallinnan keskittäminen yhteen järjestelmään johon kaikille projektiin osallistuneille olisi heti alussa luotu tunnukset. Projektissa myös todettiin Basecamp-järjestelmän sopimattomuus Scrum-projektin alustaksi sen vaatimattomien ominaisuuksien vuoksi.

Insinöörityötä kirjoittaessa projektin kolmesta vaiheesta kaksi oli saatu valmiiksi, joten projekti ei ole vielä lopussa, mutta sivuston uudistuksesta on kuitenkin suurin osa jo valmiina eikä projektin parissa työskentele enää aktiivisesti kuin muutama henkilö.

Lähteet

1 Glad, Juho. Katsomon uudistuksen konseptidokumentti. Palmu Inc., 2011.

2 Rautanen, Tommi. 2011. Software Designer, Cybercom, Helsinki. Keskustelu 28.9.2011.

3 Rautanen, Tommi. 2008. TV-opassivuston julkaisualusta. Insinöörityö. EVTEK-ammattikorkeakoulu.

4 Newell, Michael & Grashina, Maria. The Project Management Question and Answer Book. New York, Amacom, 2004.

5 Charvat, Jason. Project Management Methodologies: Selecting, Implementing, and Supporting Methodologies for Projects. Hoboken, John Wiley & Sons, 2003.

6 McConnel, Steve. Rapid Development: Taming Wild Software Schedules. Washington, Microsoft Press, 1996.

7 Cohen, D., Lindvall, M. & Costa, P. An Itroduction To Agile Methods. Verkkodokumentti. <http://www.cs.umd.edu/~mvz/cmsc435-s09/pdf/agile-paper.pdf>. Luettu 16.10.2011.

8 Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. Agile software development methods: Review and analysis. Oulu, VTT, 2002.

9 Introduction To Scrum Methodology. Verkkodokumentti. <http://www.scrummethodology.org/>. Luettu 15.10.2011.

10 Schwaber, Ken. Agile Project Management with Scrum. Microsoft Press, 2004.

11 Piipari, Markus. 2011. Projektipäällikkö, MTV3, Helsinki. Keskustelu 5.10.2011.

12 Use burn down charts in your project management reports. Verkkodokumentti. <http://chandoo.org/wp/2009/07/21/burn-down-charts/>. Luettu 16.10.2011.

13 Reilly, John. Rapid Prototyping: Moving To Business-Centric Development. Boston, Thomson Computer Press, 1996.

14 Rapid Application Development. Verkkodokumentti. <http://wsilfi.staff.gunadarma.ac.id/Downloads/files/1037/RapidApplicationDevelopment.pdf>. Luettu 30.10.2011.

15 Issue tracking system. Verkkodokumentti. <http://en.wikipedia.org/wiki/Issue_tracking_system>. Luettu 15.10.2011.

16 Atlassian – Jira issue and project tracking. Verkkodokumentti. <http://www.atlassian.com/software/jira/>. Luettu 25.9.2011.

17 Atlassian – Greenhopper. Verkkodokumentti. <http://www.atlassian.com/software/greenhopper/>. Luettu 25.9.2011.

18 Project management software, online collaboration: Basecamp. Verkkodokumentti. <http://basecamphq.com/tour/>. Luettu 29.10.2011.

Videotaulukko, 1. prototyyppi

Videotaulukko, 2. prototyyppi