jamk.fi

Qt Quick käyttöliittymäprototyypin toteutuksessa

Jani Kortelahti

Opinnäytetyö
Tammikuu 2014

Mediatekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

KUVAILULEHTI

Tekijä(t) KORTELAHTI, Jani	Julkaisun laji Opinnäytetyö	Päivämäärä 07.01.2014
	Sivumäärä 52	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty (X)

Työn nimi
Qt Quick käyttöliittymäprototyypin toteutuksessa

Koulutusohjelma
Mediatekniikan koulutusohjelma

Työn ohjaaja(t)
MANNINEN, Pasi

Toimeksiantaja(t)
Valtra Oy Ab

Tiivistelmä

Opinnäytetyön toimeksiantaja oli Valtra Oy Ab, joka valmistaa, kehittää, markkinoi ja huoltaa Valtra-traktoreita. Yritys on markkinajohtaja Pohjoismaissa ja sen traktoreita myydään yli 75 maassa. Traktoreita valmistetaan Suolahden traktoritehtaalla, jossa myös Valtran tuotekehitysyksikkö sijaitsee.

Opinnäytetyössä tutkittiin ja vertailtiin toimeksiantajalle tuttujen teknologioiden ominaisuuksia käyttöliittymäprototyyppien toteutuksessa, sekä tutustuttiin Qt Quick -teknologian tarjoamiin mahdollisuuksiin prototypoinnissa. Tavoitteena oli löytää toimeksiantajan kannalta tehokkain työkalu käyttöliittymäprototyyppien toteutukseen, sekä toteuttaa käyttöliittymäprototyyppi Qt Quick -teknologialla, jota toimeksiantaja voi käyttää teknologiaan tutustumisen ja laajemman prototyypin pohjana.

Opinnäytetyön tietoperustassa esitellään prototypointiprosessin vaiheet, sekä tutkitaan mitä asioita on yleisesti otettava huomioon prototyyppityökalua valittaessa. Lisäksi esitellään toimeksiantajalle tuttujen teknologioiden, HTML5 ja Flash, erityispiirteitä prototypoinnissa. Opinnäytetyössä esitellään tarkemmin Qt Quick -teknologiaa, ja käytännön esimerkkien avulla havainnollistetaan sen hyödyllisiä ominaisuuksia prototyyppikehityksen kannalta.

Tuloksena saatiin käyttöliittymäprototyyppi, jonka avulla todettiin Qt Quick -teknologian soveltuvan hyvin ketterään prototypointiin. Lisäksi tietoperustan pohjalta muodostettiin toimeksiantajalle teknologiasuositus tulevaisuuden prototyyppikehitystä varten.

Avainsanat (asiasanat)
Käyttöliittymäprototyyppi, prototyyppityökalut, Qt Quick

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
KORTELAHTI, Jani	Bachelor´s Thesis	07.01.2014
	Pages 52	Language Finnish
		Permission for web publication (X)

Title
Qt Quick in user interface prototyping

Degree Programme
Media Engineering

Tutor(s)
MANNINEN, Pasi

Assigned by
Valtra Oy Ab

Abstract

This Bachelor's thesis was conducted as an assignment for Valtra Oy Ab, which develops, manufactures, markets and services Valtra tractors. Valtra is the leading tractor manufacturer in the Nordic countries and its tractors are being sold in over 75 countries worldwide. Valtra tractors are built in Suolahti, where Valtra's engineering center is also located.

The objective of this Bachelor's thesis was to research and compare some of the UI prototyping technologies already known to Valtra, and also research what new opportunities Qt Quick technology could offer in UI prototyping. The objective was to find the most efficient UI prototyping tool for the client's needs and also implement a working Qt Quick UI prototype for Valtra employees to study the technology and later expand for their needs.

The theoretical part of this thesis explains prototyping as a process and also clarifies what should be taken into account when choosing a prototyping tool. Technologies already known to Valtra such as HTML5 and Flash are being assessed as prototyping tools. The thesis focuses on Qt Quick in more detail, and its best features in prototyping are introduced through practical examples.

The result of this thesis was a Qt Quick UI prototype, which proved the technology to be suitable for rapid prototyping. Additionally, the technology to be used in future UI prototyping at Valtra was suggested based on theoretical research.

Keywords
User interface, prototype, prototyping tools, Qt Quick

Miscellaneous

Sisältö

Kuviot

Taulukot

Termit

Alustariippumaton sovellus

Tarkoittaa sovellusta, jota ei ole sidottu mihinkään tiettyyn laitteistoalustaan tai käyttöjärjestelmään. Sama sovellus on siis mahdollista kääntää eri käyttöjärjestelmille ja ajaa sitä esimerkiksi Windows- ja Android-käyttöjärjestelmissä.

Deklaratiivinen ohjelmointi

Ohjelmoija kuvaa mitä ohjelma tekee, ei sitä, miten se sen tekee. Ohjelmoija voi esimerkiksi kuvailla miltä jonkin komponentin tulisi näyttää ja ohjelmointikieli hoitaa sen, miten se tulee toteuttaa.

ECMAScript

Standardoitu skriptikieli, johon esimerkiksi JavaScript ja ActionScript perustuvat. Käytetään paljon erityisesti web-selainohjelmoinnissa.

Funktio

Funktioita kutsutaan usein myös aliohjelmiksi ja niiden tarkoituksena on suorittaa jokin tietty tehtävä. Useat eri funktiot muodostavat sovelluksen toiminnallisuuden. Funktioiden avulla voidaan helposti lisätä ohjelman modulaarisuutta.

Imperatiivinen ohjelmointi

Ohjelmoija kuvaa tarkasti vaihe vaiheelta miten ohjelma pääsee haluttuun lopputulokseen.

Integroitu ohjelmointiympäristö (IDE)

Ohjelmointiympäristö johon on integroitu kaikki tarvittavat työkalut ohjelmistokehitykseen. Yleensä sisältää vähintään koodieditorin, debuggerin ja sovelluksen rakennustyökalut.

Käyttöliittymäkomponentti

Käyttöliittymäkomponentti on graafisen käyttöliittymän elementti, jolla on jokin tietty tarkoitus sovelluksessa. Tällaisia elementtejä ovat esimerkiksi painikkeet, tekstikentät ja liukupalkit.

Olio-ohjelmointi

Olio-ohjelma koostuu useista eri olioista, jotka sisältävät tietoa ja toiminnallisuutta. Oliot kommunikoivat keskenään ja lähettävät tietoa toisilleen. Olio-ohjelmoinnin avulla suuriakin ohjelmia on helpompi hallita ja ylläpitää, sillä jokaisella oliolla on oma vastuunsa ja tehtävänsä.

Prototyyppi

Tässä opinnäytetyössä termillä prototyyppi tarkoitetaan käyttöliittymäprototyyppiä. Prototyypillä tarkoitetaan mallia lopputuotteesta.

Vektorigrafiikka

Vektorigrafiikkaa voidaan skaalata ilman että kuvasta tulee rakeinen, koska objektien muodot ja ominaisuudet kuvataan koordinaatein ja matemaattisin funktioin.

WYSIWYG-editori

Lyhenne tulee sanoista *”What You See Is What You Get”.* WYSIWYG-editorit ovat yleensä hyvin aloittelijaystävällisiä, sillä ne näyttävät reaaliaikaisesti miltä työ tulee lopulta näyttämään. WYSIWYG-editoreissa tulee yleensä mukana erilaisia drag-and-drop-työkaluja, tai muita pikavalintoja, joiden avulla käyttöliittymien luonti on nopeaa.

1 Työn lähtökohdat

Käyttöliittymäprototyypin toteutus on tärkeä osa käyttöliittymän kehitystyötä. Prototyypin avulla lopputuotteen ideaa on mahdollista visualisoida ja havainnollistaa niin, että jo kehitystyön alkuvaiheessa on mahdollista saada arvokasta palautetta niin asiakkailta kuin muiltakin käyttäjiltä. Tällä tavalla on mahdollista helpottaa lopputuotteen kehitystyötä ja säästää projektin loppuvaiheen resursseja.

Mitä asioita on otettava huomioon ennen prototyypin toteutuksen aloittamista? Millainen prototyyppi on projektin kannalta hyödyllisin, ja miten se olisi viisainta toteuttaa? Prototyypin toteutukseen ei kuitenkaan haluta sitoa liikaa resursseja, eikä sen muokkaamisen ja kehittämisen tulisi olla hankalaa. Voiko prototyypistä saada konkreettista hyötyä myös lopputuotteen toteutukseen ja jos voi, onko se järkevää?

Tämän opinnäytetyön tavoitteena oli löytää työn toimeksiantajalle, Valtra Oy:lle, heidän tarkoituksiinsa sopivin tapa toteuttaa käyttöliittymäprototyyppejä. Opinnäytetyön alussa tutkittiin, mitä asioita on otettava huomioon prototyyppityökalua valittaessa, jo ennen prototypoinnin aloittamista. Työssä esitellään Valtralle jo tuttujen teknologioiden erityispiirteitä sekä käydään tarkemmin läpi alustariippumattoman Qt Quick -teknologian mahdollisuuksia prototyyppikehityksessä. Opinnäytetyössä keskityttiin muita teknologioita tarkemmin Qt Quick -teknologiaan, koska sen tarjoamat mahdollisuudet prototypoinnissa kiinnostivat toimeksiantajaa erityisesti. Yhteenvedossa vertailtiin eri prototyyppityökalujen ominaisuuksia ja työvaiheita sekä arvioitiin niiden soveltuvuutta ketterään prototypointiin.

Valtra Oy Ab valmistaa, kehittää, markkinoi ja huoltaa Valtra-traktoreita. Yritys on markkinajohtaja Pohjoismaissa, mutta sen traktoreita myydään yli 75 maassa. Se on valmistanut traktoreita jo vuodesta 1951. Traktoreita valmistetaan Suolahden traktoritehtaalla sekä Brasiliassa Mogi das Cruezesissa. Suolahden tehdasalueella toimivat kokoonpanotehdas, voimansiirtotehdas, varaosa- ja huoltokeskus sekä asiakaspalvelukeskus Atrium. Myös Valtran hallinto ja tuotekehitysyksikkö sijaitsevat Suolahdes-

sa, traktoritehtaan alueella. Valtralla on noin 2300 työntekijää eri puolilla maailmaa, ja se valmistaa vuosittain noin 18 000 traktoria. (Tietoa Valtrasta 2013.)

Valtra on yksi yhdysvaltalaisen AGCO-yhtymän kansainvälisistä tuotemerkeistä. AGCO on yksi maailman suurimmista maatalouskoneita suunnittelevista, valmistavista ja myyvistä yrityksistä. AGCO:n valikoimaan kuuluu täysi valikoima erilaisia maatalouskoneita useilta eri tuotemerkeiltä, ja sen tuotteita myy 3900 jälleenmyyjää yli 140 maassa. AGCO Suomi Oy myy Suomessa Valtran lisäksi sisarmerkkejä Fendtiä ja Challengeria. (Mt.)

Valtran tuotekehitys vastaa kaikkien Valtran valmistamien traktoreiden suunnittelusta ja kehityksestä. Uusi tuotekehityskeskus on avattu vuonna 2007 ja sen laboratorioihin on rakennettu mm. kaiuton testihuone, jäähdytyshuone, dynamometrejä ja monipuolinen runkotestipenkki. Kaikkien traktoreiden lopullinen testaus toteutetaan todellisissa työolosuhteissa, jotta voidaan taata käyttövarmuus vaativimmissakin olosuhteissa ja tilanteissa. (Mt.)

2 Käyttöliittymäprototyypin määrittely

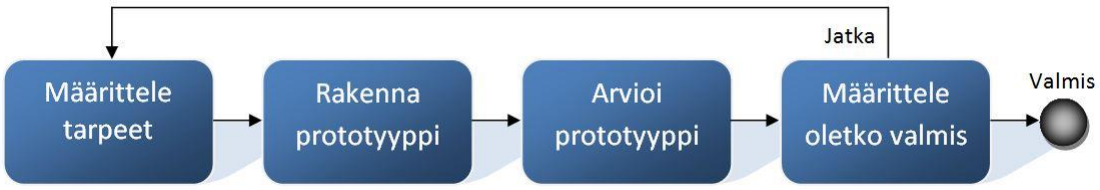
2.1 Käsite ja työprosessi

Kun puhutaan prototyypistä, sillä tarkoitetaan yleensä jonkinlaista mallia lopputuotteesta. Nykyään prototyypistä puhuttaessa jo käsitteestä usein ymmärretään prototypoinnin iteratiivinen luonne. Prototyypin tekeminen lopputuotteesta usein varmistaa konseptin toimivuutta, ja sen avulla voidaan helposti testata ohjelmistolle asetettuja vaatimuksia käytännössä. (Arnowitz, Arent & Berger 2006, 3-4.)

Hix ja Hartson (1993) toteavat kirjassaan, että prototyyppi on oiva keino testata käyttöliittymän graafisen designin, käytettävyyden ja interaktiivisuuden yhteen toimivuutta jo ohjelmiston kehitysprosessin alussa. He korostavat, että ketterä prototyyppikehitys ei kuitenkaan anna syytä olla laiska tai huolimaton alkuperäisen designin kanssa. Ketterä, iterointiin perustuva prototyyppikehitys tulee kysymykseen siksi, että on mahdotonta valmistaa täysin toimiva prototyyppi heti ensimmäisellä kerralla, pelkkien design ohjeiden perusteella. (Hix & Hartson 1993.)

Onkin selvää, että esimerkiksi pelkän designin esittely asiakkaille saattaa olla siitä tulevan palautteen kannalta harhaan johtavaa, sillä designista puuttuu kaikki toiminnallisuus ja interaktiivisuus. Hienon ja toimivan näköisen käyttöliittymän saa helposti toteutettua niin, että sitä on vaikeaa tai jopa epämiellyttävää käyttää. Kun designin ja interaktiivisuuden yhteen toimivuutta testataan jo varhaisessa vaiheessa, mahdolliset epäkohdat on helppo korjata ja tarvittaessa testata uudestaan asiakkailla.

Prototyypin tekeminen on siis iteratiivinen prosessi, jonka päätarkoituksena on tuottaa malli lopputuotteesta. Amblerin (2001) mukaan käyttöliittymäprototyypin toteutusprosessin voi jakaa neljään päätason vaiheeseen (ks. kuvio 1).



Kuvio 1. Prototyypin kehitysprosessi (alkup. kuvio Ambler 2001)

Ensimmäisessä vaiheessa on tärkeää määritellä, miksi prototyyppi tehdään. Amblerin (2001) mukaan ensimmäisessä vaiheessa määritellään käyttäjän tarpeet, mutta jo alkuvaiheessa prototyypin tekoa ohjaavat myös muut kuin käyttäjän tarpeet. Eri tarpeiden selvittäminen auttaa myös selventämään motivaatioita prototyypin toteutukseen.

Arnowitz ja muut (2006, 9) listaavat teoksessaan neljä eri syytä tai motivaatiota prototyypin toteutukseen:

- Tuottomotivaatio - Halu kehittää menestyvä tuote markkinoille järkevässä aikataulussa
- Paine pysyä edellä kilpailijoita
- Riski siitä, että ohjelmisto ei olekaan käyttäjien mielestä haluttava, kun se julkaistaan markkinoille
- Loppukäyttäjä ei välttämättä halua sellaista tuotetta, mikä on tuottavaa tai muuten mahdollista yrityksen tuottaa.

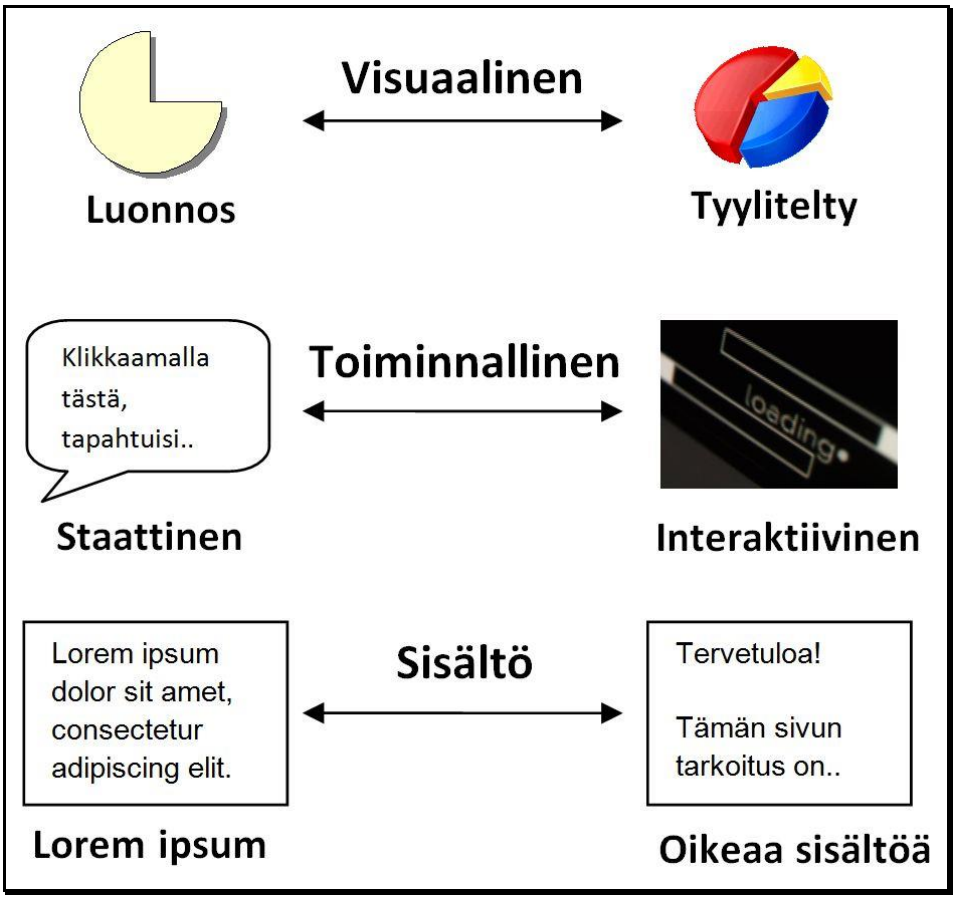
Motivaatioiden ja käyttäjätarpeiden selvitys myös määrittää pitkälti tulevan prototyypin ja sitä kautta myös lopputuotteen vaatimukset. Osa motivaatioista voi olla ristiriidassa toistensa kanssa, kuten joissain tapauksissa esimerkiksi yrityksen tuottomotivaatiot ja aikataulupaineet vastaan käyttäjien odotukset. Tämän vuoksi prototyypit ovat oiva apukeino kultaisen keskitien löytämiseen, mikä tyydyttäisi kaikkia osapuolia.

2.2 Prototyypin tarkkuustaso

Se, milloin prototyyppi on valmis ja milloin voidaan lopettaa kehitysprosessin iterointi, riippuu paljon siitä, kuinka tarkkaa prototyyppiä ollaan kehittämässä. Prototyypit voidaan yleisesti jakaa kolmeen tarkkuusluokkaan: matalan, keskitason ja korkean tarkkuuden prototyyppeihin.

Tarkkuustasolla tarkoitetaan sitä, kuinka viimeistellyn näköinen ja käytettävyydeltään hiottu prototyyppi on asiakkaille ja muille käyttäjille. Sillä ei ole mitään tekemistä sen kanssa, kuinka prototyyppi on toteutettu, esimerkiksi kuinka viimeisteltyä sen koodi on. Matalan tarkkuuden prototyypit ovat vain korkean tason abstrakteja kuvauksia suunnitellusta designista tai esimerkiksi layoutista. Tämän tason prototyypit ovat hyvä vaihtoehto esimerkiksi, jos graafisia yksityiskohtia ei ole vielä lyöty lukkoon tai jos ne tulevat todennäköisesti muuttumaan. Korkean tarkkuuden prototyypit taas kuvaavat yksityiskohtaisesti ohjelmiston ulkoasun lisäksi myös toiminnalliset osat ja interaktiivisuuden. Tämän tason prototyyppejä voidaan käyttää jo lopputuotteen tarkkana spesifikaationa. Keskitason tarkkuuden prototyypit sijoittuvat näiden välille ja tulevat monesti kysymykseen, jos resurssit eivät riitä prototyypin kehittämiseen matalan tarkkuuden prototyypistä aina korkeimpaan tarkkuuteen. (Hartson & Pyla 2012.)

Prototyypin tarkkuustasoa voidaan määritellä myös tarkemmin. Cerejo (2010) jakaa prototyypin tarkkuuden kolmeen osaan: visuaaliseen, toiminnalliseen ja sisällölliseen tarkkuuteen (ks. kuvio 2). Valitsemalla jokaiselle osa-alueelle omaan käyttötarkoitukseen sopivan tarkkuustason voidaan helpottaa prototyypin toteutuksen ja sisällön suunnittelua. (Cerejo 2010.)



Kuvio 2. Prototyypin tarkkuustasot (alkup. kuva Cerejo 2010)

Cerejon (2010) mukaan projektin alkuvaiheessa ei kannata panostaa liikaa prototyypin visuaaliseen näyttävyyteen, sillä se saattaa ohjata testikäyttäjiä kiinnittämään liikaa huomiota pelkästään ulkomuotoon. Projektin edetessä voi alkaa lisätä visuaalisia yksityiskohtia ja tätä kautta visuaalista tarkkuutta. Toiminnallisessa tarkkuudessa staattinen tarkkuustaso riittää lähinnä esittelemään sovelluksen ideaa, kun taas interaktiivista prototyyppiä voisi käyttää esimerkiksi käytettävyystestauksissa. Sisällönkään osalta ei ole välttämättä järkevää heti alussa täyttää prototyyppiä oikealla sisällöllä, sillä se saattaa viedä testikäyttäjien huomion pois olennaisesta. Kun prototyyppi alkaa olla hiotumpi, voi olla hyvä korvata keksitty sisältö oikealla ja testata, vaikuttaako se prototyypin yleiskuvaan ja designiin. (Mt.)

3 Prototyyppityökalut

3.1 Prototyyppityökalun valinta

Ennen käyttöliittymäprototyypin toteutuksen aloitusta on tärkeää kartoittaa oikeat työvälineet prototyypin toteutusta varten. Prototyyppityökalun valinta voi joskus olla vaikeaa, sillä vaihtoehtoja on niin paljon. On olemassa työkaluja/sovelluksia, jotka on varta vasten kehitetty prototyyppien kehitykseen, mutta ei pidä unohtaa, että prototyyppejä voi tehdä myös kynällä ja paperilla. Seuraavassa on lueteltuna muutama tunnettu prototyyppityökalu:

- Microsoft Word
- Microsoft PowerPoint
- Microsoft Visio
- Adobe Flash
- Adobe Dreamweaver
- Adobe Photoshop
- Adobe Illustrator
- Kynä/paperi
- Qt Creator
- Microsoft SketchFlow
- InVision
- Fluid UI.

Työkalua valittaessa kannattaa muistaa, että prototyypin toteutuksessa voi käyttää useampaakin työkalua. Esimerkiksi alussa voi olla hyvä luonnostella ulkoasua kynällä paperille, sen jälkeen tehdä rautalankamallit Visiolla ja kuvakäsikirjoitus PowerPointilla. Prototyyppi olisi kaikkein parasta toteuttaa työkalulla, joka on jo ennestään tuttu tekijälle. Jos prototyyppiä toteutetaan useammalla työkalulla, on otettava huomioon se, että kaikki työkalut ovat yhteensopivia keskenään tai että niistä saadaan ulos

tiedosto sellaisessa formaatissa, joka käy kaikkiin prototyyppityökaluihin. (Arnowitz ym. 2006, 157–158.)

Työkalun valintaan vaikuttavat myös prototyypin haluttu tarkkuustaso sekä se, kuinka pitkää elinikää prototyypiltä odotetaan. Prototyypin kehityksessä puhutaan yleensä joko pois-heitto-prototyypeistä (throwaway prototyping) tai evolutiivisista prototyypeistä (evolutionary prototyping). Nimensä mukaisesti pois-heitto-prototyypit ovat vain vaiheita kohti lopputuotetta, ja ne heitetään pois ennen lopputuotteen toteuttamista, kun taas evolutiiviset prototyypit on tarkoitus kehittää lopputuotteeksi saakka. Jos päätetään tehdä evolutiivinen prototyyppi, on erittäin tärkeää, että prototyyppiä testataan tarkasti koko ohjelmistokehitystyön ajan. (Sharp, Rogers & Preece 2007.)

Hix ja Hartson (1993) toteavat, että pois-heittotyylistä prototyypin kehitystä käytetään yleensä, jos prototyypin ja lopputuotteen alustat poikkeavat toisistaan. Vaikka tämä on vielä tänäkin päivänä varmasti yleistä, on kuitenkin huomioitava, että nykyään on tarjolla paljon alustariippumattomia sovelluskehitysympäristöjä, joiden mukana tulevat helposti käytettävät drag-and-drop-työkalut soveltuvat erittäin hyvin sovellusprototyyppien tekemiseen. Tällaisissa prototyypeissä on se etu, että niitä pystyy ajamaan useilla eri alustoilla.

Kaksi merkittävintä asiaa, jotka vaikuttavat työkalun valintaan, ovat prototyypin haluttu tarkkuustaso sekä se, tehdäänkö pois-heitto vai evolutiivista prototyyppiä. Jos päätetään tehdä karkeantason luonnosprototyyppi, joka heitetään pois käyttäjätestausten jälkeen, ei kannata valita työkalua, joka vaatii paljon uuden opettelua tai paljon ohjelmointia. Tässä tapauksessa kannattaa valita sellainen työkalu, joka on ennestään kaikkein tutuin ja jolla muutosten teko ja kehitystyö on kaikkein nopeinta.

Jos päätetään tehdä korkean tarkkuuden prototyyppi, joka kehitetään lopputuotteeksi tai joka tulee toimimaan lopputuotteen tarkkana spesifikaationa, valinnassa kannattaa jo varhaisessa vaiheessa ottaa huomioon lopputuotteen alusta. Jos prototyyppi täytyy toteuttaa jollain eri teknologialla kuin lopputuote, on muistettava, että

prototyypistä saatava hyöty lopputuotteeseen rajoittuu lähinnä grafiikoihin ja spesifikaationa olemiseen.

Hartson ja Pyla (2012) listaavat teoksessaan kahdeksan ominaisuutta, jotka löytyisivät täydellisestä prototyyppityökalusta:

- Nopea ja vaivaton tehdä muutoksia
- Helppo myös ohjelmointitaidottoman käyttää ja muokata prototyypin design ominaisuuksia
- Sisäänrakennettuna yleisiä interaktiotoimintoja ja mahdollisuus hakea näitä lisää plug-in-kirjastoista
- Tuki erilaisille laitteille ja eri tavoille käyttää niitä, esimerkiksi kosketusnäytöille, puheentunnistukselle ja eleille
- Helppo tehdä ja muokata prototyypissä navigointia erilaisten painikkeiden ja valikoiden avulla
- Kommunikointi ulkoisten prosessien ja ohjelmien kanssa toiminnallisuuden lisäämiseksi sekä esimerkiksi datan siirron helppous
- Mahdollisuus tuoda tekstiä, grafiikkaa ja muita medioita eri lähteistä
- Mahdollisuus viedä komponentteja lopputuotteeseen.

3.2 Adobe Flash Professional

Flash Professional on työkalu, jonka avulla käyttöliittymäprototyyppien tekeminen on kohtuullisen nopeaa ja vaivatonta. Se soveltuu myös erinomaisesti erilaisten animaatioiden tai vaikkapa pelien tekemiseen. Flash on alustariippumaton työkalu, mutta Flash-sovellusten ajaminen vaatii joko Adobe AIR tai Flash Player -ajoympäristön kohdelaitteessa. (Flash Professional CC / Features 2013.)

Flash Professional kuuluu nykyään Adoben Creative Cloud -palveluun. Creative Cloud on palvelu, johon käyttäjä voi ostaa kuukausimaksuun perustuvan jäsenyyden, johon kuuluuvat kaikki Adoben ammattikäyttöön tarkoitetut luovan työn tietokonesovel-

lukset. Tämän lisäksi jäsenyyteen kuuluvat ohjelmistot pysyvät päivitettyinä ja niitä pystyy käyttämään usealta eri tietokoneelta. (Adobe Creative Cloud 2013.)

Flashin vahvuudet prototyyppityökaluna ovat sen aikajana (timeline) ja näyttämö (stage), joiden avulla on erittäin helppoa ja nopeaa tehdä grafiikkaa ja luoda interaktiivisuutta. Näyttämölle voi piirtää grafiikkaa erilaisilla työkaluilla sekä raahata erilaisia muotoja tai jopa valmiita käyttöliittymäkomponentteja Flashin omasta komponenttikirjastosta. Aikajanan avulla voidaan esimerkiksi prototyypin eri sivut jakaa eri vaiheisiin (frame) aikajanalla, ja navigointi prototyypissä tapahtuu liikkumalla näissä eri frameissa. Käyttöliittymä on mahdollista piirtää ja toteuttaa kokonaan Flash Professionalilla ja toiminnallisuuksien sekä navigoinnin ohjelmointi toteutetaan ActionScript 3.0 -ohjelmointikielellä, jota varten ohjelmistosta löytyy myös monipuolinen editori.

ActionScript 3.0 on oliopohjainen ohjelmointikieli, joka perustuu ECMAScriptiin, kansainvälisesti standardoituun komentosarjakieleen (Grossman & Huang 2006). Tämän vuoksi ActionScript muistuttaakin toista hyvin suosittua komentosarjakieltä JavaScriptiä, joka myös perustuu kyseiseen standardiin.

Adobe tarjoaa Creative Suite -paketissaan loistavat työkalut luovaan työhön. Jos projektin graafinen suunnittelija työskentelee esimerkiksi Photoshopilla tai Illustratorilla, on hyvin houkuttelevaa tehdä prototyyppiin interaktiivisuutta saman tuoteperheen Flash Professionalilla. Adoben ehdoton vahvuus onkin eri sovellusten yhteen toimivuus. Samanaikainen työskentely eri sovellusten välillä toimii todella saumattomasti, ja esimerkiksi Illustratorilla tehdyt vektorigrafiikat pysyvät hyvin muokattavina myös Flash Professionalin puolella lisättyine efekteineen ja muine ominaisuuksineen.

Creative Cloud -palvelu maksaa yritykselle 69,99 € (ei sis. ALV) käyttäjää kohden edellyttäen vuosisitoumusta kuukausilaskutuksella. Myös yksittäisen sovelluksen ostaminen on mahdollista, jolloin suositushinta samoilla ehdoilla on 29,99 € (ei sis. ALV) kuukaudessa käyttäjää kohden. (Adobe Creative Cloud / Jäsenyysohjelmat 2013.)

Seuraavassa taulukossa on listattuna Flash-teknologian erityispiirteitä prototypoinnissa (ks. taulukko 1).

Taulukko 1. Flash-teknologia prototypoinnissa

+	-
Aikajana	Vaatii Flash-ajoympäristön kohdelaitteessa.
Grafiikkatyökalut	Vähän ulkopuolisia UI-komponenttikirjastoja
Omien animaatioiden teko	Flash-yhteisö on pienentynyt HTML5 buumin vuoksi.
Yhteensopivuus Adoben graafisiin ohjelmistoihin (Photoshop, Illustrator).	
Kehitysympäristö: Adobe Flash Professional **Ohjelmointikieli**: ActionScript 3.0	

3.3 HTML5 teknologiat

HTML-prototyypit ovat nostaneet suosiotaan erityisesti sen jälkeen, kun HTML-kielen uusin versio HTML5 kasvatti suosiotaan. Vaikka sillä virallisesti tarkoitetaan vain HTML-sivujen kuvauskielen uusinta versiota, se on vakiintunut käsittämään kaikki uudet nettisovellukset, joissa hyödynnetään myös muita läheisiä teknologioita, kuten JavaScriptiä ja CSS3:a. HTML5:sta onkin luvattu teknologiaa, joka muuttaisi nettisivut sovelluksiksi, jotka toimisivat kaikissa päätelaitteissa ja selaimissa. Kolehmaisen artikkelin mukaan sen on odotettu jopa korvaavan mobiililaitteiden natiivisovellukset. (Kolehmainen 2012, 7.)

HTML:n avulla kuvataan web-sivun rakennetta ja mm. sitä, mitkä sanat muodostavat otsikon, mitkä kuuluvat kappaleeseen ja mitkä tarvitsevat luettelomerkin eteensä. Sen avulla myös määritellään eri sivujen väliset linkitykset, kuvat, videot ja lomak-

keet. CSS:n avulla määritellään sivuston ulkoasu, esimerkiksi kirjasintyyppi, taustavärit ja taustakuvat. Lisäksi sen avulla voidaan määritellä eri elementtien paikat sivulla. (Larsen 2013.)

JavaScriptin avulla web-sivuista saa tehtyä interaktiivisia niin, että ne reagoivat käyttäjän toimiin. Sillä voi myös antaa sivulle ja sen elementeille erilaisia visuaalisia efektejä. JavaScript pohjautuu ECMAScriptiin, ja sen syntaksissa on samankaltaisuuksia muihin ohjelmointikieliin, kuten Javaan, C++:aan ja C:hen. (Pollock 2013.)

Jos tehdään matalan tarkkuuden karkeaa prototyyppiä, jo pelkästään HTML/CSS–yhdistelmällä on mahdollista luoda havainnollistavia navigoitavia prototyyppejä, joissa esimerkiksi sovelluksen layoutin esitteleminen onnistuu mainiosti. Jos prototyyppiin halutaan lisää interaktiivisuutta, se on helpointa toteuttaa JavaScriptillä niin, että ottaa avuksi jonkin JavaScript-kirjaston. JavaScript-kirjastoissa on valmiina lukuisia funktioita ja objekteja, jotka nopeuttavat toiminnallisuuksien ja efektien luomista. Lisäksi monia kirjastoja on mahdollisuus laajentaa entisestään erilaisten lisäosien kautta.

Prototyyppikehityksen kannalta erityisesti UI-kirjastot ovat hyödyllisiä, sillä niistä löytyy myös valmiita, muokattavia käyttöliittymäkomponentteja. Yksi suosituimmista, eniten käytetyistä ja monipuolisimmista kirjastoista on jQuery ja käyttöliittymäsuunnittelun ja kehityksen kannalta hyödyllinen jQuery UI. JQuery UI on rakennettu jQuery JavaScript-kirjaston päälle, ja se sisältää käyttöliittymäinteraktioita, efektejä, komponentteja sekä erilaisia teemoja (jQuery user interface 2013).

HTML-kehityksessä ei ole mitään standardisoituja tai suositeltuja sovelluskehitysympäristöjä, joten oikean ja sopivan työkalun löytäminen jää kehittäjän harteille. Tämä saattaa osoittautua yllättävän vaikeaksi, jos ei ole aikaisempaa kokemusta ohjelmoinnista tai sovelluskehitysympäristöistä. Pelkällä tekstieditorilla kehittäminen käy sitä työläämmäksi, mitä tarkempaa ja laajempaa prototyyppiä ollaan kehittämässä. Joissain kehitysympäristöissä tulee mukana drag-and-drop työkaluja, joilla luonnostelu onnistuu hyvin, mutta tällaiset työkalut saattavat jättää vain vähän mahdollisuuk-

sia personoinnille, eivätkä tällä tavalla toteutetut sivut välttämättä tue haluttuja JavaScript-kirjastoja.

HTML-prototyypin yksi vahvuus on siinä, että sen saa varsin vaivattomasti suurellekin joukolle testattavaksi internetin välityksellä. Testaajan ei tarvitse asentaa koneelleen erikseen mitään ohjelmia tai ajoympäristöjä, kunhan koneesta löytyy internet-selain. Jos prototyypin haluaa julkaista tällä tavalla suurelle joukolle testattavaksi, on kuitenkin varmistettava sen toimivuus eri selaimissa. Tämä taas saattaa viedä jonkin verran resursseja pois prototyypin kannalta olennaisista asioista.

Seuraavassa taulukossa on yhteenvetona HTML5-teknologian erityispiirteitä prototypoinnissa (ks. taulukko 2).

Taulukko 2. HTML5-teknologia prototypoinnissa

+	-
Paljon UI-komponenttikirjastoja (myös muita JavaScript-kirjastoja). Nopeuttaa kehittäjän työtä huomattavasti.	Ei grafiikan piirtotyökaluja (CSS-määrittelyillä voi toteuttaa grafiikoita).
Prototyyppi on helppo julkaista suurelle yleisölle.	Prototyyppi tulee optimoida eri ajoympäristöille, eli eri web-selaimille, jos se julkaistaan suurelle yleisölle.
HTML5 buumin myötä kehittäjäyhteisö on erittäin laaja ja aktiivinen.	Ei kovin ketterä muutosten teossa
HTML-sivuihin on tarvittaessa helppoa upottaa esimerkiksi Flash-sisältöä.	
Kehitysympäristö: Vaihtoehtoja on lukuisia **Ohjelmointikieli**: JavaScript	

4 Qt ja Qt Quick

4.1 Qt

Qt on alustariippumaton ohjelmisto- ja käyttöliittymäkehitysympäristö. Se tukee kaikkia keskeisiä työpöytäkäyttöjärjestelmiä, kuten Windows, Mac OS X ja Linux, sekä sulautettujen järjestelmien käyttöjärjestelmiä, kuten sulautettu Linux ja Windows Embedded. Näiden lisäksi se tukee käytetyimpiä sulautettujen laitteiden reaaliaikaisia käyttöjärjestelmiä, kuten VxWorks, QNX Neutrino ja Integrity, sekä suosituimpia mobiilikäyttöjärjestelmiä, kuten Android, iOS, BlackBerry ja SailFish. (Digia julkaisi Qt 5.1 2013.)

Qt:n juuret juontavat vuoteen 1994, jolloin Eirik Chambe-Eng ja Haavard Nord perustivat norjalaisen Trolltech-yrityksen, jonka tuote Qt oli. Kesäkuussa 2008 Nokia osti Qt:n Trolltechiltä, mutta maaliskuussa 2011 Digia osti Qt:n kaupallisen lisensointi- ja palveluliiketoiminnan Nokialta. Tämän kaupan myötä Digialle siirtyi noin 3500 työpöytä- ja sulautettujen ohjelmistojen asiakasyritystä eri toimialoilta. Syyskuussa 2012 Digia osti koko Qt-kehitysympäristön Nokialta ja otti samalla vastuulleen kaikki Qt-teknologiaan liittyvät toiminnot, kuten tuotekehityksen, sekä kaupallisen ja avoimen lähdekoodin lisensointi- ja palveluliiketoiminnan. (About us 2013; Digia ostaa Qt-ohjelmistojen kaupallisen lisensointi- ja palveluliiketoiminnan Nokialta 2011; Digia ostaa koko Qt-kehitysympäristön Nokialta 2012.)

Muihin alustariippumattomiin ohjelmistokehitysympäristöihin verrattuna Qt sovellukset ovat kohdelaitteissa erittäin suorituskykyisiä, sillä ne ovat natiiveja sovelluksia. Qt tukee tällä hetkellä yli 14:ta käyttöjärjestelmää monissa eri suoritinarkkitehtuureissa ja yli 500 000 kehittäjää maailmanlaajuisesti käyttää sitä. (Digia julkaisi Qt 5.1 2013; Uusi mobiiliympäristöön suunnattu Qt-tuote tehostaa sovelluskehitystä 2013.)

Opinnäytetyössä tutkittiin Qt:n versiota 5.1, jonka merkittävimpiin uudistuksiin kuului tuki Android- ja iOS-laitteille Technology Preview statuksella, sekä uusia moduulei-

ta kuten Qt Sensors. Sensors-moduulin avulla pääsee käsiksi mobiililaitteiden sensorien dataan, sekä liike-eleiden tunnistukseen QML:n ja C++ rajapintojen kautta. Tämän lisäksi Qt 5.1:ssä oli yli 3000 parannusta verrattuna aiempaan Qt versioon. (Qt 5.1 available 2013; What's new in Qt 5.1 2013.)

Qt:n uusin versio 5.2 julkaistiin joulukuussa 2013, samaan aikaan Qt Mobile Edition:in kanssa. Qt Mobile Edition on uusi mobiilikehittäjille optimoitu kehitysympäristö. Qt 5.2 toi mukanaan mm. täyden tuen Android-, iOS- ja Blackberry 10 -laitteille, sekä uuden JavaScript-virtuaalikoneen, joka on optimoitu QML ja Qt Quick käyttöön. Lisäksi julkaistiin paljon pienempiä parannuksia, sekä mm. uusia mobiililaitteille optimoituja käyttöliittymäkomponentteja. (Qt 5.2 Beta Available 2013; Uusi mobiiliympäristöön suunnattu Qt-tuote tehostaa sovelluskehitystä 2013.)

4.2 Qt Quick ja QML

4.2.1 Yleistä

Qt Quick on moderni käyttöliittymäkehitykseen tarkoitettu teknologia, joka erottelee deklaratiivisen käyttöliittymän toteutuksen ja imperatiivisen ohjelmointilogiikan toisistaan. Käyttöliittymäkerros toteutetaan deklaratiivisella QML-kielellä, joka muistuttaa hyvin paljon CSS- ja JavaScript-kieliä. Vaikka käyttöliittymäkerros toteutetaan QML:llä, Qt Quick -sovelluksilla on vahva C++ tausta, jonka johdosta QML-kerroksesta saa suoran natiivin yhteyden laitteeseen. Ohjelmistologiikka ja datayhteydet voidaan kirjoittaa erikseen Qt C++ ohjelmointirajapinnoilla. (Qt Quick 2013.)

QML on suunniteltu kaventamaan designereiden ja ohjelmistokehittäjien välistä kuilua, sillä sen avulla on kohtalaisen vaivatonta luoda modernin näköisiä käyttöliittymiä. Sen avulla esimerkiksi erilaisten graafisten efektien ja animaatioiden lisääminen sovellukseen onnistuu helposti. Muita sisäänrakennettuja ominaisuuksia ovat mm. 3D grafiikoiden, OpenGL varjostimien, partikkeleiden ja multi-touch eleiden tuki. Tämän lisäksi eri elementeille on helppoa luoda erilaisia tiloja, esimerkiksi painikkeelle

voi oletustilan lisäksi luoda ”painettu”-tilan, jolloin esimerkiksi painikkeen grafiikat muuttuvat. Tällaiset ominaisuudet nopeuttavat käyttöliittymien luontia ja ovat erityisen hyödyllisiä esimerkiksi prototypoinnissa. (Mt.)

Qt 5.1 toi Qt Quick:iin kaksi erityisesti prototypoinnin kannalta hyödyllistä moduulia, Controls ja Layouts. Controls-moduuli sisältää mm. useita eri käyttöliittymäkomponentteja, kuten painikkeita, valintaruutuja, liukupalkkeja ja tekstikenttiä. Komponentit näyttävät ja käyttäytyvät työpöytäsovelluksissa kuten vastaavat natiivit komponentit, mutta niiden ulkoasua on helppoa myös muokata sisäänrakennetun Styling API:n avulla. Layouts-moduulia taas käytetään elementtien asemointiin ja sijoitteluun näytöllä. (Qt 5.1 available 2013.)

4.2.2 JavaScriptin käyttö QML:ssä

JavaScriptiä voi käyttää QML:ssä neljällä eri tavalla. Sen avulla voidaan esimerkiksi määritellä QML-elementtien ominaisuuksia ja signaalin käsittelijöitä. Myös omien JavaScript-funktioiden sisällyttäminen suoraan QML-elementteihin on mahdollista. Tämän lisäksi QML-tiedostoihin voi tuoda omia ulkoisia JavaScript-tiedostoja, joiden funktioihin ja muuttujiin pääsee tämän jälkeen käsiksi QML:stä. (JavaScript Expressions in QML Documents 2013.)

Elementtien ominaisuuksien arvojen asettaminen JavaScriptin avulla onnistuu, kunhan JavaScript-lausekkeen lopputulos on arvo, joka voidaan sijoittaa elementin ominaisuuden arvoksi (mt.). Yleisin tapa sitoa JavaScript-lauseke elementin arvoon, ilmenee seuraavasta kuviosta (ks. kuvio 3).

```
1   import QtQuick 2.0
2
3   Item {
4       width: 300
5       height: 300
6
7       Rectangle {
8           id: random
9           anchors.centerIn: parent
10          width: Math.random() * 100
11          height: 60
12          color: width < 80 ? "red" : "blue"
13      }
14
15  }
```

Kuvio 3. Ominaisuuden arvon asettaminen JavaScriptillä

Kuvion 3 esimerkissä suorakulmion leveys on satunnainen luku 0-100 välillä (rivi 10). Jos suorakulmion leveys on pienempi kuin 80px sen väri on punainen, muussa tapauksessa se on sininen (rivi 12). Tällaisessa tapauksessa suorakulmion leveyden arvo generoidaan kun elementti luodaan. Jos leveyttä kuitenkin muutetaan elementin luonnin jälkeen jollain tavalla, suorakulmion väri tulee muuttumaan uuden leveyden mukaisesti.

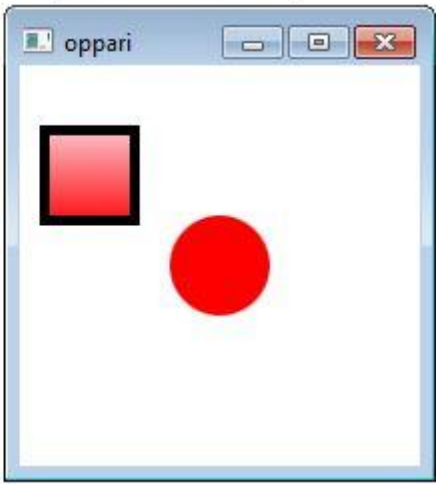
4.2.3 Visuaaliset elementit

Kaikki Qt Quick:in visuaaliset elementit periytyvät Item-luokasta. Se tarjoaa kaikille visuaalisille elementeille useita perusominaisuuksia, kuten esimerkiksi opacity ja transform. Opacityn avulla voidaan määritellä elementin läpinäkyvyyttä, kun taas transformien avulla elementtejä voidaan mm. skaalata tai kääntää. Muutokset näihin ominaisuuksiin voidaan myös animoida, joka lisää sulavuutta käyttöliittymäelementtien ominaisuuksien muutoksiin. (Use Case – Visual Elements in QML 2013.)

Tavallisimmat graafiset elementit saadaan QML:ssä toteutettua Rectangle-elementtien avulla. Rectangle-elementit ovat nimensäkin mukaisesti suorakulmioita ja niitä voidaan mm. värittää eri väreillä tai liukuväreillä, antaa reunuksia ja pyöristää

kulmista. Kulmia pyöristämällä esimerkiksi ympyröiden luominen Rectangle-elementeistä onnistuu helposti (ks. kuvio 4). (Mt.)

```
1  import QtQuick 2.0
2
3  Rectangle {
4      id: background
5      width: 200
6      height: 200
7
8      Rectangle {
9          id: absoluteRect
10         x:10; y: 30
11         width: 50; height: 50
12         border { width: 5; color: "black" }
13         gradient: Gradient {
14             GradientStop { position: 0.0; color: "pink" }
15             GradientStop { position: 1.0; color: "red" }
16         }
17     }
18
19     Rectangle {
20         id: relativeRect
21         anchors.centerIn: background
22         width: 50; height: 50
23         color: "red"
24         radius: height / 2
25     }
26 }
```



Kuvio 4. Rectangle-elementti

Toinen hyvin paljon käytetty elementti on Image-elementti, jonka avulla voidaan näyttää erilaisia kuvia. Image-elementin source-ominaisuuden avulla voidaan määritellä missä kuva sijaitsee, esimerkiksi kuva internetissä tai kuva projektin resurssitiedostoissa. (Mt.)

4.2.4 Elementtien asemointi

QML:ssä elementtejä on helppo asetella säännöllisesti erilaisten asemointielementtien (positioner) avulla. Tällaisia valmiita elementtejä ovat mm. Column, Row, Grid ja

Flow elementit. Niiden avulla elementtien järjestely sarakkeisiin, riveihin ja ruudukoihin on helppoa. Flow käyttäytyy muuten hyvin samalla tavalla kuin Grid, mutta se järjestää lapsielementtinsä ensin horisontaalisti ja sitten vertikaalisti, kunnes kukin rivi on täynnä. Tämän vuoksi, jos lapsielementit ovat erikokoisia, ne eivät välttämättä ole samassa linjassa vertikaalisti. Flow:ta käytetäänkin erityisesti esimerkiksi sanojen asetteluun. (Item positioners 2013.)

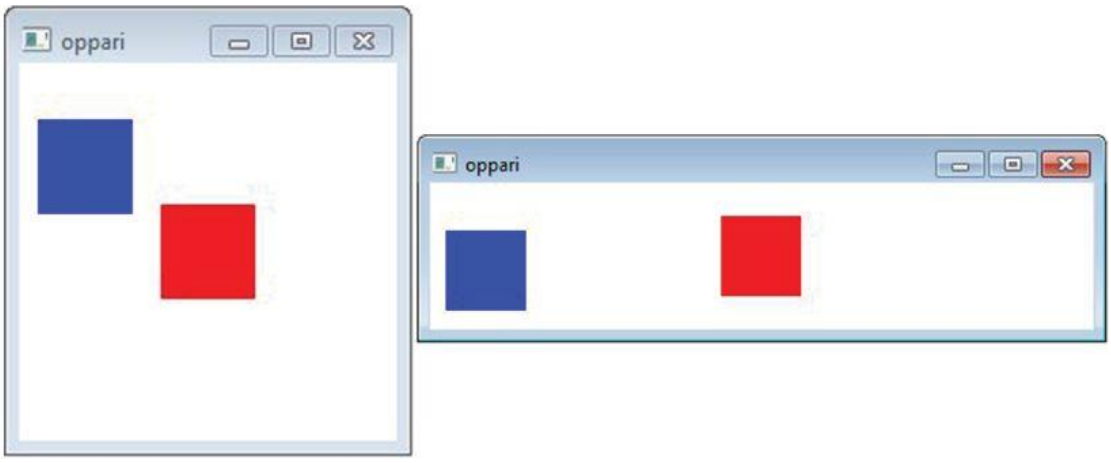
Qt 5.1:n mukana tullut Qt Quick Layouts -moduuli on tarkoitettu erityisesti responsiivisten layouttien luomiseen. Se tarjoaa valmiita layout-tyylejä, kuten RowLayout, ColumnLayout ja GridLayout, joiden avulla voidaan järjestää komponentteja näytöllä riveihin, sarakkeisiin tai ruudukoihin, mutta asettelun lisäksi nämä layout-elementit osaavat tarvittaessa myös muokata omien lapsielementtiensä kokoa.
(Qt Quick Layouts 2013.)

QML:ssä on asemointielementtien ja Qt Quick Layouttien lisäksi muitakin keinoja asemoida elementtejä. Elementit voidaan sijoittaa näytölle myös manuaalisesti, asettamalla elementille x- ja y-koordinaatit. Nämä annetut koordinaatit ovat suhteessa elementin vanhemman vasempaan yläkulmaan. Elementin x- ja y-koordinaatit voidaan myös sitoa riippuvaisiksi esimerkiksi jonkin toisen elementin ominaisuuksista (ks. kuvio 5). (Use Case - Positioners and Layouts in QML 2013.)

```
1  import QtQuick 2.0
2
3  Rectangle {
4      id: background
5      width: 200
6      height: 200
7
8      Rectangle {
9          id: absoluteRect
10         x:10
11         y: 30
12         width: 50; height: 50
13         color: "blue"
14     }
15
16     Rectangle {
17         id: relativeRect
18         x: (background.width / 2) - (relativeRect.width / 2)
19         y: (background.height / 2) - (relativeRect.height / 2)
20         width: 50; height: 50
21         color: "red"
22     }
23 }
24
```

Kuvio 5. Elementin manuaalinen sijoittelu

Kuvion 5 koodiesimerkissä on taustalla 200 x 200 pikselin kokoinen neliö, jonka sisällä on kaksi pienempää neliötä, toisen sijainti on määritelty absoluuttisesti ja toisen sijainti vaihtuu dynaamisesti aina jos taustaneliön kokoa muuttaa. Neliö, jonka id on relativeRect, keskittyy aina keskelle taustaneliötä, mutta sininen neliö on aina vasemmalta 10 px ja ylhäältä 30 px (ks. kuvio 6).



Kuvio 6. Staattinen ja dynaaminen elementti

Toinen vaihtoehto on käyttää ns. ankkureita, joilla elementit voi ankkuroida toisiin elementteihin. Jokaisella elementillä on kuusi ankkurilinjaa: left, right, vertical center, top, bottom ja horizontal center. Vertikaalit ankkurilinjat voi ankkuroida toisen elementin vastaaviin vertikaaleihin ankkurilinjoihin ja horisontaalit ankkurilinjat toimivat samalla periaatteella. (Mt.)

Ankkuroinnilla toteutettuna esimerkiksi elementin keskittäminen toisen elementin sisään onnistuu huomattavasti helpommin, kuin kuvion 5 esimerkissä (ks. kuvio 7).

```
1  import QtQuick 2.0
2
3  Rectangle {
4      id: background
5      width: 200
6      height: 200
7
8      Rectangle {
9          id: absoluteRect
10         x:10
11         y: 30
12         width: 50; height: 50
13         color: "blue"
14     }
15
16     Rectangle {
17         id: relativeRect
18         anchors.centerIn: background
19         width: 50; height: 50
20         color: "red"
21     }
22 }
```

Kuvio 7. Ankkurointi elementin keskelle

4.2.5 Animaatiot

QML on suunniteltu erityisesti sulavasti toimivien modernien käyttöliittymien tekemiseen. Sen vuoksi erilaisten elementtien siirtymiä tai ominaisuuksien muutoksia voidaan animoida helposti niin, että elementit eivät vain katoa paikasta A ja ilmesty paikkaan B. Tämän sijasta elementin sijainti vaihtuu sulavasti ja sen sijainti näytetään

koko matkan ajan. Samalla tavalla esimerkiksi muutokset elementin läpinäkyvyydessä voidaan animoida. (Usecase – Animation in QML 2013.)

Animaatiot voidaan liittää elementtien eri ominaisuuksien arvojen vaihtumiseen, tai esimerkiksi elementtien tilojen vaihtumisen yhteyteen. Lisäksi animaatioita voidaan määritellä yleisemmälläkin tasolla niin, että kohde elementti ja sen animoitavat ominaisuudet määritellään vasta animaatiossa ja se ajetaan esimerkiksi hiiren klikkauksen jälkeen. (Mt.)

Jos animaatio liitetään esimerkiksi elementin y-akselin sijainnin muutokseen, animaatio ajetaan aina kun elementin sijainti vaihtuu y-akselilla, riippumatta siitä, mikä aiheuttaa sijainnin muutoksen. Seuraavassa kuviossa esimerkki tällaisesta animaatiosta (ks. kuvio 8).

```
1   import QtQuick 2.0
2
3   Item {
4       width: 400
5       height: 400
6
7       Rectangle {
8           id: circle
9           color: "blue"
10          width: 120
11          height: 120
12          radius: height / 2
13
14          MouseArea {
15              anchors.fill: parent
16              // siirretään ympyrää y-akselilla
17              onClicked: circle.y = 280
18          }
19
20          // Aina kun ympyrän sijainti y-akselilla muuttuu,
21          // tämä animaatio ajetaan
22          Behavior on y {
23
24              NumberAnimation {
25                  duration: 1200
26                  easing.type: Easing.OutElastic
27              }
28          }
29      }
30  }
31
```

Kuvio 8. Ominaisuuden muutoksen animointi

Yksi helppo ja paljon käytetty tapa lisätä liikettä ja interaktiivisuutta elementteihin on QML:n State-toiminnallisuus. Uuteen tilaan on helppo listata kaikki ominaisuudet uusine arvoineen, joiden halutaan muuttuvan oletustilasta. Yleensä eri tilat määritellään juurielementissä. Jos tilan vaihdon yhteydessä halutaan aina ajaa animaatio, se onnistuu helpoiten Transitions-toiminnallisuuden avulla (ks. kuvio 9). (Mt.)

```
import QtQuick 2.0

Item {
    id: container
    width: 400
    height: 400

    Rectangle {
        id: circle
        color: "blue"
        width: 120
        height: 120
        radius: height / 2

        MouseArea {
            anchors.fill: parent
            // vaihdetaan ympyrän tila
            onClicked: {
                if(container.state == "invisible") {
                    container.state = ""
                } else {
                    container.state = "invicible"
                }
            }
        }
    }

    states: [

        State {
            name: "invisible"
            PropertyChanges {
                target: circle
                opacity: 0
            }
        }
    ]

    transitions: [
        Transition {
            // Animaatio ajetaan aina kun containerin
            // tila muuttuu
            NumberAnimation { properties: "opacity" }
        }
    ]
}
```

Kuvio 9. Animaatio tilan vaihtuessa

Animaatioita ei ole aina pakko sitoa elementin tilan tai ominaisuuksien vaihdoksiin, ne voidaan myös määritellä yleisemmin. Tällöin suoraan animaatiossa voidaan määri-

tellä esimerkiksi elementin sijainnin muutokset (ks. kuvio 10) (mt.). Kuvion 10 animaatiossa keltainen neliö liukuu x-akselilla kohdasta 0 kohtaan 200 puolessa sekunnissa ja tämän jälkeen takaisin alkuun samassa ajassa.

```
1  import QtQuick 2.0
2
3  Item {
4      width: 400
5      height: 400
6
7      Rectangle {
8          id: square
9          color: "yellow"
10         width: 120
11         height: 120
12
13         MouseArea {
14             anchors.fill: parent
15             // animaatio alkaa kun neliötä painetaan
16             onClicked: animation.running = true;
17         }
18     }
19
20     // Tämä animaatio on kohdennettu suorakulmiolle,
21     // jonka id on square
22     SequentialAnimation {
23         id: animation
24
25         NumberAnimation {
26             target: square
27             property: "x"
28             from: 0
29             to: 200
30             duration: 500
31         }
32
33         NumberAnimation {
34             target: square
35             property: "x"
36             from: 200
37             to: 0
38             duration: 500
39         }
40     }
41 }
42
```

Kuvio 10. Sijainnin muutoksen määrittely animaatiossa

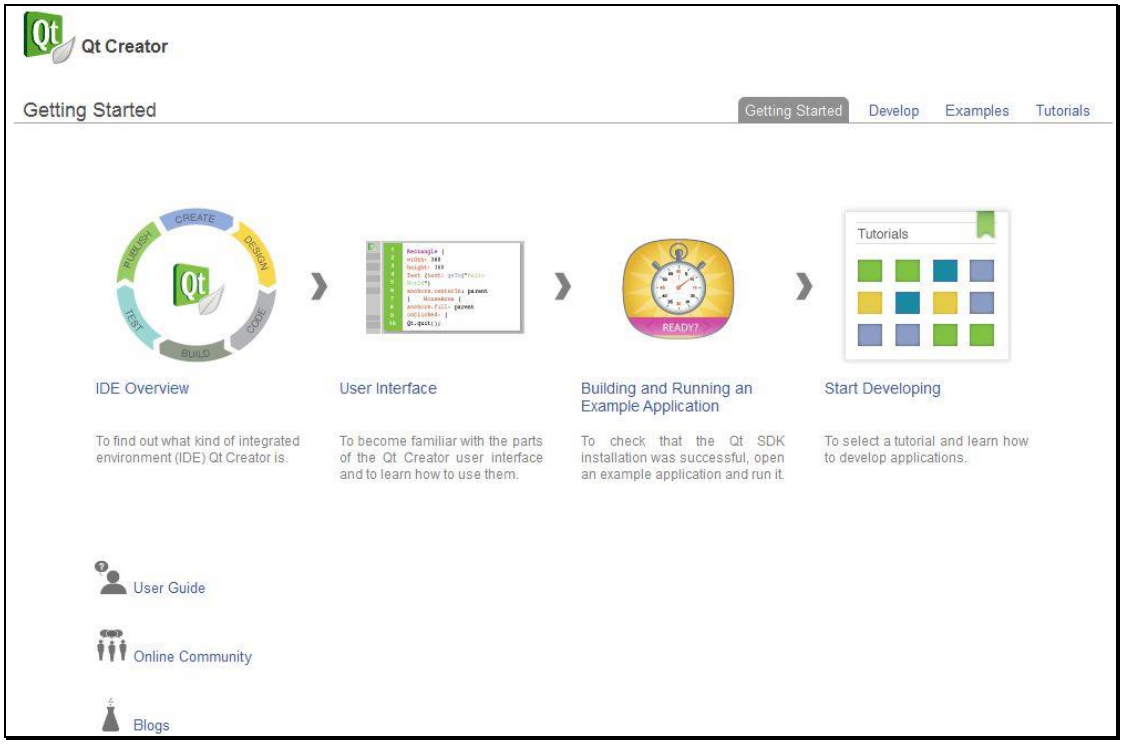
5 Qt Quick -teknologia käytännössä

5.1 Qt Creator

Qt Creator on integroitu ohjelmointiympäristö IDE (integrated development environment), joka on optimoitu Qt -sovellusten suunnitteluun ja kehitykseen. Sen avulla voidaan suunnitella sovelluksen graafista ulkoasua, ohjelmoida ja julkaista sovellus useille eri kohdealustoille. Lisäksi se tarjoaa työkalut virheiden etsimiseen ja poistamiseen, sekä ohjelmointikoodin analysointiin. (IDE Overview 2013.)

Qt Creator toimii Windows, Linux/X11 ja Mac OS X työpöytäkäyttöjärjestelmissä. Sen koodieditori tukee C++, JavaScript ja QML-kieliä ja se ymmärtää nämä kielet koodina, ei pelkkänä tekstinä. Tämän vuoksi se osaa tarjota kontekstiin sopivaa apua, sekä mm. täydentää kirjoitettua koodia. Lisäksi siitä löytyy tuki versionhallinalle, sillä siihen voidaan integroida jokin ulkoinen versionhallintajärjestelmä kuten Git, Subversion tai Bazaar. (Qt Creator IDE and tools 2013.)

Kun Qt Creator:in käynnistää ensimmäisen kerran, näytölle avautuu erittäin aloittelijaystävällinen tervetuloa-näyttö (ks. kuvio 11). Näytön kautta pääsee tutustumaan Qt Creatorin ominaisuuksiin, selailemaan valmiita esimerkkisovelluksia tai katsomaan opastevideoita. Videoiden joukosta löytyy lyhyitä käytännön läheisiä opastevideoita, sekä mm. kuvattuja esityksiä Qt Developer Days tapahtumista, jotka saattavat kestää yli tunninkin. Aloitusnäytön kautta löytyy myös suorat linkit Qt -projektin yhteisösivulle, sekä Qt:n viralliselle blogisivulle.



Kuvio 11. Qt Creator:in aloitusnäyttö

Uuden projektin luominen onnistuu helposti ohjatun projektin luonnin avulla. Ohjatussa luonnissa käyttäjältä kysytään askel askeleelta, millainen projekti luodaan ja Qt Creator luo kaikki tarvittavat tiedostot ja määrittelee asetukset käyttäjän tekemien valintojen mukaan. Ohjatun projektin luonnin avulla voidaan luoda mm. Qt Widget-, Qt Quick- ja HTML5-sovelluksia. (Creating Projects 2013.)

5.2 Omien UI-komponenttien luominen

QML tarjoaa paljon valmiita uudelleen käytettäviä käyttöliittymäkomponentteja Qt Quick Controls -moduulin kautta, mutta niitä voi tehdä helposti myös itse. Omia elementtityyppejä tai komponentteja voi tehdä uuteen QML-tiedostoon. Jos halutaan esimerkiksi tehdä oma painikekomponentti, luodaan ensin uusi QML-tiedosto Creatorin yläpalkin valikosta File – Create new file or project – QML File (QT Quick 2) ja annetaan QML-tiedostolle nimeksi esimerkiksi MyButton.

MyButton.qml tiedostossa on luonnin jälkeen automaattisesti yksi Rectangle-elementti. Tähän tiedostoon määritellään MyButton-elementin ominaisuudet ja graafinen ulkoasu. Graafisen ulkoasun voi määritellä joko QML-kielellä, tai käyttää siinä jotain valmista kuvatiedostoa. MyButton-elementin ulkoasu on määritelty QML:llä ja se koostuu kolmesta eri elementistä. Rectangle muodostaa elementin taustan, Text sisältää painikkeen tekstin ja MouseArea on klikattava alue (ks. kuvio 12).

```
1  import QtQuick 2.0
2
3  Rectangle {
4      id: button
5      width: 200
6      height: 62
7      radius: 10
8      border { width: 3; color: "#d1d1d1" }
9
10     gradient: Gradient {
11         GradientStop {
12             position: 0
13             color: "#fefefe"
14         }
15
16         GradientStop {
17             position: 1
18             color: "#b8bac6"
19         }
20     }
21
22     Text {
23         id: label
24         font.pixelSize: 14
25         anchors.centerIn: button
26         text: "Click here"
27     }
28
29     MouseArea {
30         anchors.fill: parent
31     }
32 }
33
```

Kuvio 12. MyButton-komponentti

Tällä tavoin toteutettu MyButton-painike on nyt helppoa lisätä käyttöliittymään. Kun elementin lisää johonkin toiseen QML-tiedostoon, sille voi luonnin yhteydessä määritellä sen juurielementin ominaisuuksia, kuten esimerkiksi width, height, radius, x , y jne. MyButton-elementin juurielementti on taustan muodostava Rectangle, jonka

lapsielementit ovat Text ja MouseArea. Luonnin yhteydessä ei pysty suoraan viittaamaan lapsielementtien ominaisuuksiin, kuten esimerkiksi tekstielementtiin. Tämän vuoksi jokaisessa MyButton-elementissä lukee sama teksti, ”Click here”. Jos elementin luonnin yhteydessä halutaan määritellä lapsielementin ominaisuuksia, näistä ominaisuuksista tulee määritellä property-ominaisuus juurielementissä. Seuraavassa kuviossa, rivillä 6, on määritelty label property, joka viittaa MyButton:in tekstielementin tekstiominaisuuteen (ks. kuvio 13.)

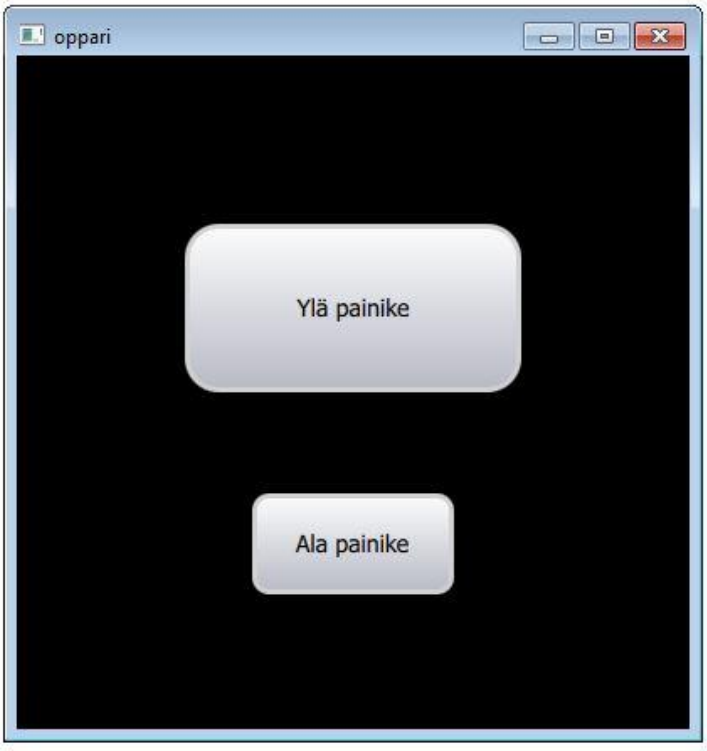
```
1  import QtQuick 2.0
2
3  Rectangle {
4      id: button
5
6      property alias label: label.text
7
8      width: 200
9      height: 62
10     radius: 10
11     border { width: 3; color: "#d1d1d1" }
12
13     gradient: Gradient {
14         GradientStop {
15             position: 0
16             color: "#fefefe"
17         }
18
19         GradientStop {
20             position: 1
21             color: "#b8bac6"
22         }
23     }
24
25     Text {
26         id: label
27         font.pixelSize: 14
28         anchors.centerIn: button
29         text: "Click here"
30     }
31
32     MouseArea {
33         anchors.fill: parent
34     }
35 }
36
```

Kuvio 13. Elementin property-ominaisuus

Property:n avulla on mahdollista MyButton-elementin luonnin yhteydessä antaa painikkeelle oma yksilöllinen teksti. Seuraavassa kuviossa on luotu mustan taustan pääl-

le kaksi erikokoista MyButton-elementtiä, joissa molemmissa lukee eri teksti (ks. kuvio 14).

```
import QtQuick 2.0

Rectangle {
    width: 400
    height: 400
    color: "black"

    MyButton {
        width: 200
        height: 100
        radius: 20
        x: 100
        y: 100
        label: "Ylä painike"
    }

    MyButton {
        width: 120
        height: 60
        radius: 10
        anchors.horizontalCenter: parent.horizontalCenter
        y: 260
        label: "Ala painike"
    }
}
```



Kuvio 14. Oman painikkeen käyttö

5.3 Signaalit ja signaalinkäsittelijät

QML:ssä eri komponentit kommunikoivat keskenään signaalien ja signaalinkäsittelijöiden avulla. Monilla valmiilla komponenteilla on joitain signaaleja ja signaalinkäsittelijöitä jo valmiiksi, mutta niitä voi tehdä myös itse. Esimerkiksi MouseArea-elementillä on clicked-signaali, joka lähetetään aina kun klikataan MouseArea-alueen sisäpuolella. Tämä signaali lähettää mukanaan mouse nimisen parametrin, joka sisältää tarkempaa tietoa hiiren klikkauksesta, kuten esimerkiksi hiiren x- ja y-koordinaatit. Signaalin ottaa vastaan onClicked-signaalinkäsittelijä, jossa voidaan esimerkiksi JavaScriptin avulla määritellä halutut tapahtumat ja esimerkiksi tulostaa näytölle, missä kohtaa hiirtä klikattiin. Jokaisella luodulla signaalilla on aina automaattisesti oma on-alkuinen signaalinkäsittelijä. (Signal and handler event system 2013.)

Edellisen esimerkin MyButton-elementti ei sisältänyt mitään toiminnallisuutta. Seuraavassa esimerkissä lisätään MyButton-elementtiin oma activated-signaali rivillä 7 ja lähetetään se rivillä 35, kun käyttäjä klikkaa painikkeen MouseArea:sta (ks. kuvio 15).

```
import QtQuick 2.0

Rectangle {
    id: button

    property alias label: label.text
    signal activated(real mouseX, real mouseY)

    width: 200
    height: 62
    radius: 10
    border { width: 3; color: "#d1d1d1" }

    gradient: Gradient {
        GradientStop {
            position: 0
            color: "#fefefe"
        }

        GradientStop {
            position: 1
            color: "#b8bac6"
        }
    }

    Text {
        id: label
        font.pixelSize: 14
        anchors.centerIn: button
        text: "Click here"
    }

    MouseArea {
        anchors.fill: parent
        onClicked: button.activated(mouse.x, mouse.y)
    }
}

```

Kuvio 15. Signaali MyButtonissa

Tällä tavalla voidaan elementin luonnin yhteydessä määritellä jokaisen MyButton-elementin signaalinkäsittelijän toiminnot. Koska activated-signaali lähettää mukanaan hiiren klikkauspaikan x- ja y-koordinaatit, signaalinkäsittelijässä voidaan esimerkiksi tulostaa nämä tiedot (ks. kuvio 16).

```
import QtQuick 2.0

Rectangle {
    width: 400
    height: 400
    color: "black"

    MyButton {
        width: 200
        height: 100
        radius: 20
        x: 100
        y: 100
        label: "Ylä painike"
        onActivated: console.log("x:"+mouseX+" y:"+mouseY)
    }
}

```

Kuvio 16. Signaalinkäsittelijä MyButtonissa

Signaaleja voi yhdistää myös toisiin signaaleihin tai esimerkiksi suoraan johonkin JavaScript-funktioon connect() -metodin avulla. Jos signaalin yhdistää suoraan funktioon, funktio suoritetaan automaattisesti kun signaali lähetetään. Yhdistämällä signaalin toiseen signaaliin, voidaan muodostaa erilaisia signaaliketjuja ja saada aikaan useampia tapahtumia samanaikaisesti (ks. kuvio 17). (Signal and handler event system 2013.)

```
import QtQuick 2.0

Rectangle {
    id: bg

    signal sendMessage

    onSendMessage: console.log("Message 2")

    MouseArea {
        id: mousearea
        anchors.fill: parent
        onClicked: console.log("Message 1")
    }

    Component.onCompleted: mousearea.clicked.connect(sendMessage)
}

```

Kuvio 17. Signaaliketju (mt. muokattu)

Kuvion 17 esimerkissä kun MouseArea:n clicked-signaali lähetetetään, myös sendMessage-signaali lähetetään. Tämä tulostaisi konsoliin: Message 1 Message 2.

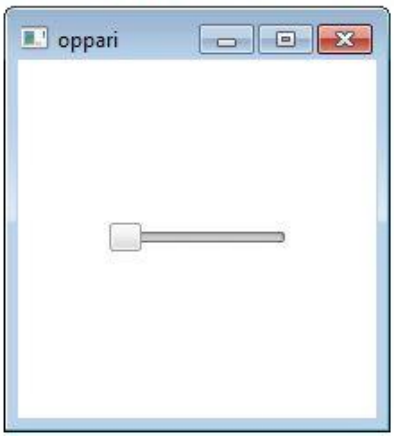
5.4 Valmiiden UI-komponenttien tyylittely

Qt Quick Controls -moduuli sisältää lukuisia valmiita UI-komponentteja, joiden avulla käyttöliittymien kokoaminen käy nopeasti ja esimerkiksi prototypointi nopeutuu. Valmiit komponentit ovat ulkoasultaan työpöytäkomponenttien näköisiä, joten personoallisia sovelluksia tehdessä komponenttien ulkoasua tulee pystyä muokkaamaan. Valmiiden komponenttien tyylittely onnistuu Qt Quick Controls Styles -alamoduulin avulla.

Tämän hetkisen Controls Styles -moduulin versio on 1.0 ja se sisältää tyylittelyluokat ScrollViewStyle, TabViewStyle ja TableViewStyle näkymille, sekä ButtonStyle, CheckBoxStyle, ComboBoxStyle, ProgressBarStyle, RadioButtonStyle, SliderStyle ja TextFieldStyle komponenteille. (Qt Quick Controls Styles 2013.)

Jos sovelluksessa halutaan käyttää valmiita komponentteja, täytyy QML-tiedostoon tuoda QtQuick Controls -moduuli, kuten kuviossa 18 rivillä 2 (ks. kuvio 18). Esimerkissä näkyy tavallinen täysin muokkaamaton Slider-komponentti.

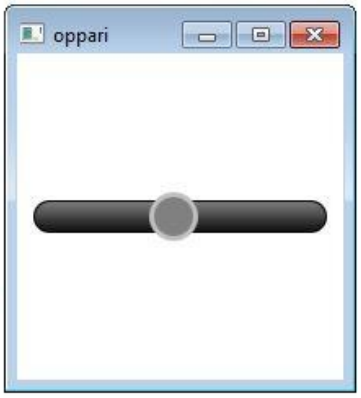
```
1  import QtQuick 2.1
2  import QtQuick.Controls 1.0
3
4  Rectangle {
5      id: bg
6      width: 200
7      height: 200
8
9      Slider {
10         anchors.centerIn: parent
11     }
12 }
13
```



Kuvio 18. Muokkaamaton Slider-komponentti

Sliderin ulkoasun muokkaaminen onnistuu SliderStyle-tyylittelyluokan avulla. Tyylittely määritykset voidaan tehdä suoraan Slider-elementin luonnin yhteydessä (ks. kuvio 19) tai ne voidaan sijoittaa erilliseen QML-tiedostoon. Tyylimääritysten tekeminen erilliseen tiedostoon on suositeltavaa, jos tyyliteltyä elementtiä on tarkoitus käyttää useassa paikassa.

```
1  import QtQuick 2.1
2  import QtQuick.Controls 1.0
3  import QtQuick.Controls.Styles 1.0
4
5  Rectangle {
6      id: bg
7      width: 200
8      height: 200
9
10     Slider {
11         anchors.centerIn: parent
12         style: SliderStyle {
13             groove: Rectangle {
14                 implicitWidth: 180
15                 implicitHeight: 20
16                 gradient: Gradient {
17                     GradientStop {
18                         color: "#7d7e7d"
19                         position: 0
20                     }
21                     GradientStop {
22                         color: "#0e0e0e"
23                         position: 1
24                     }
25                 }
26                 border.width: 1
27                 radius: 10
28             }
29             handle: Rectangle {
30                 anchors.centerIn: parent
31                 color: control.pressed ? "black" : "grey"
32                 border.width: 3
33                 border.color: "silver"
34                 width: 30
35                 height: 30
36                 radius: height / 2
37             }
38         }
39     }
40 }
41
```



Kuvio 19. Tyylitelty Slider-komponentti

SliderStyle:ssä määritellään liukupalkin taustan (groove), sekä kahvan (handle) ulkoasu. Kuvion 19 esimerkissä, rivillä 31, määritellään kahvan väri. Kahva on harmaa, mutta painettaessa sen väri muuttuu mustaksi. Tämän rivin control viittaa liukupalkkiin, johon tyyli on asetettu, sen kautta voidaan tiedustella onko liukupalkin kahva

painettuna. Tyylimäärittelyt voidaan myös siirtää erilliseen tiedostoon, esimerkiksi MySliderStyle.qml-tiedostoon (ks. kuvio 20).

```
// MySliderStyle.qml
import QtQuick 2.1
import QtQuick.Controls.Styles 1.0

SliderStyle {
    groove: Rectangle {
        implicitWidth: 180
        implicitHeight: 20
        gradient: Gradient {
            GradientStop {
                color: "#7d7e7d"
                position: 0
            }
            GradientStop {
                color: "#0e0e0e"
                position: 1
            }
        }
        border.width: 1
        radius: 10
    }
    handle: Rectangle {
        anchors.centerIn: parent
        color: control.pressed ? "black" : "grey"
        border.width: 3
        border.color: "silver"
        width: 30
        height: 30
        radius: height / 2
    }
}

```

Kuvio 20. Tyylimäärittelyt erillisessä tiedostossa

Tämän jälkeen Sliderin tyylin voi määritellä esimerkiksi main.qml-tiedostossa seuraavan esimerkin mukaisesti (ks. kuvio 21).

```
// main.qml
import QtQuick 2.1
import QtQuick.Controls 1.0

Rectangle {
    id: bg
    width: 200
    height: 200

    Slider {
        anchors.centerIn: parent
        style: MySliderStyle {}
    }
}

```

Kuvio 21. Tyylin asettaminen erillisestä tiedostosta

Jos valmiista Qt Quick -komponentista haluaa tehdä oman uudelleen käytettävän tyylitellyn komponentin, se kannattaa tehdä kokonaan omaan QML-tiedostoon ja täten luoda siitä oma komponentti (ks. kuvio 22). Kun Sliderista tekee oman komponentin, esimerkiksi MySlider, tähän tiedostoon kannattaa sijoittaa kaikki sliderin toiminnallisuudet ja muuttujat, mutta kaikki ulkoasuun vaikuttavat määrittelyt voi edelleen pitää erillisessä MySliderStyle.qml-tiedostossa. Erilaisia ulkoasutiedosto- ja/teemoja voi tehdä useammankin ja aina oman komponentin määrittelyn yhteydessä voidaan määritellä, mitä teemaa missäkin sliderissa käytetään.

```
1  // MySlider.qml
2  import QtQuick 2.1
3  import QtQuick.Controls 1.0
4  import QtQuick.Controls.Styles 1.0
5
6  Slider {
7      id: mySlider
8
9      /* SliderStyle{} on QtQuick sliderin oletustyyli */
10     property Component personalStyle: SliderStyle{}
11
12     property real sliderValue: control.value
13
14     style: personalStyle
15 }
16
```

Kuvio 22. Oma slider-komponentti Qt Quick -komponentin pohjalta

Kuvion 22 esimerkissä luodaan personalStyle-property, jonka avulla voidaan myöhemmin määritellä jokaiselle luodulle liukupalkille haluttu tyyli. Tämän MySlider-komponentin oletustyyli on Qt Quick Controls Sliderin oletustyyli. Jos halutaan luoda MySlider-komponentti, jonka ulkoasu on MySliderStyle:in mukainen, se onnistuu seuraavan kuvion osoittamalla tavalla (ks. kuvio 23).

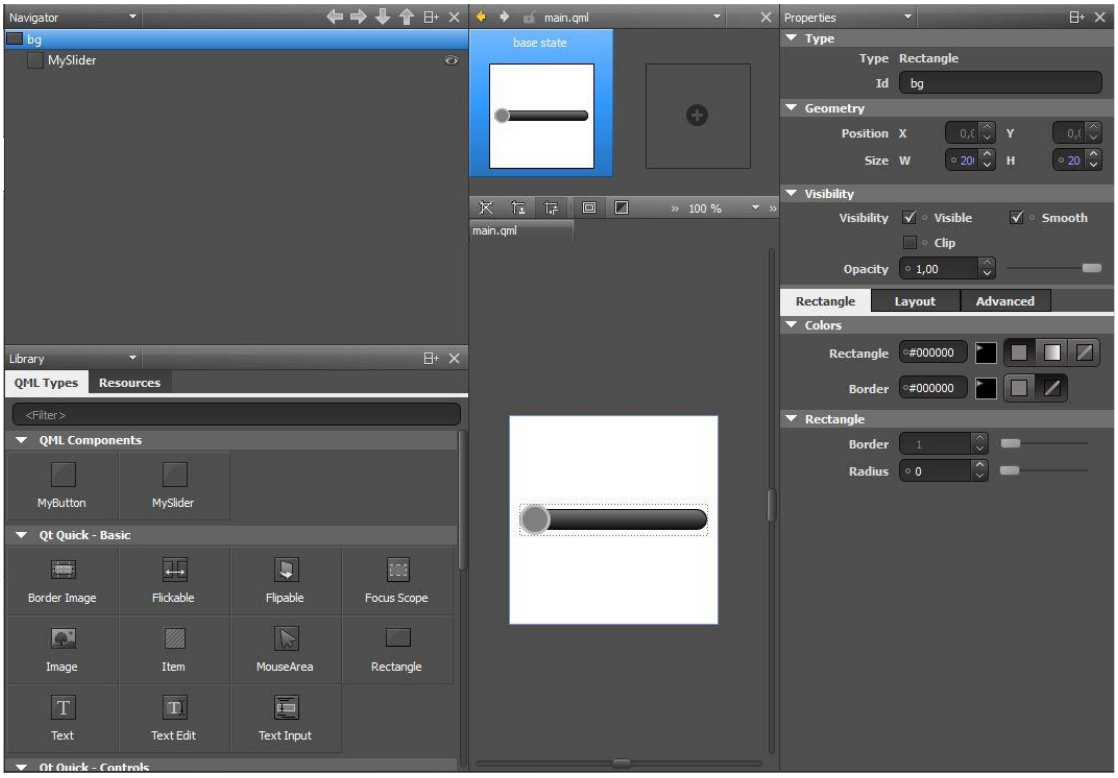
```
import QtQuick 2.1
import QtQuick.Controls 1.0

Rectangle {
    id: bg
    width: 200
    height: 200

    MySlider {
        anchors.centerIn: parent
        personalStyle: MySliderStyle{}
    }
}

```

Kuvio 23. MySlider-komponentin luonti ja tyylin määritys

5.5 Qt Quick Designer

Qt Quick Designer on Qt Creatorin mukana tuleva WYSIWYG-editori käyttöliittymien helppoon ja nopeaan rakentamiseen. Designer-editori toimii sulavasti yhteen koodieditorin kanssa, joten molempia voi käyttää sulavasti yhteen omien tarpeiden mukaisesti. Designerin avulla voi myös esikatsella QML-tiedostoja kääntämättä niitä ensin. (Qt Creator IDE and tools 2013.)

Designer työtila on jaettu viiteen pienempään ikkunaan, jotka ovat navigaattori, kirjasto, kanveesi, ominaisuudet ja tilat (ks. kuvio 24). Navigaattori-ikkunasta näkee valitun QML-tiedoston elementit puumuodostelmassa, josta myös elementtien hierarkia ilmenee. Kirjasto-ikkunassa näkee kaikki komponentit, joiden avulla käyttöliittymän voi koota. Kanveesi on designerin työskentelytila, josta näkee työn alla olevan tiedoston. Ominaisuudet-ikkunan kautta voidaan määritellä valitun komponentin ominaisuuksia, kuten sijaintia, kokoa, väriä tai vaikka reunuksia. Tilat-ikkunassa voidaan luoda juurielementille uusia tiloja (state) ja tehdä niihin haluttuja muutoksia.



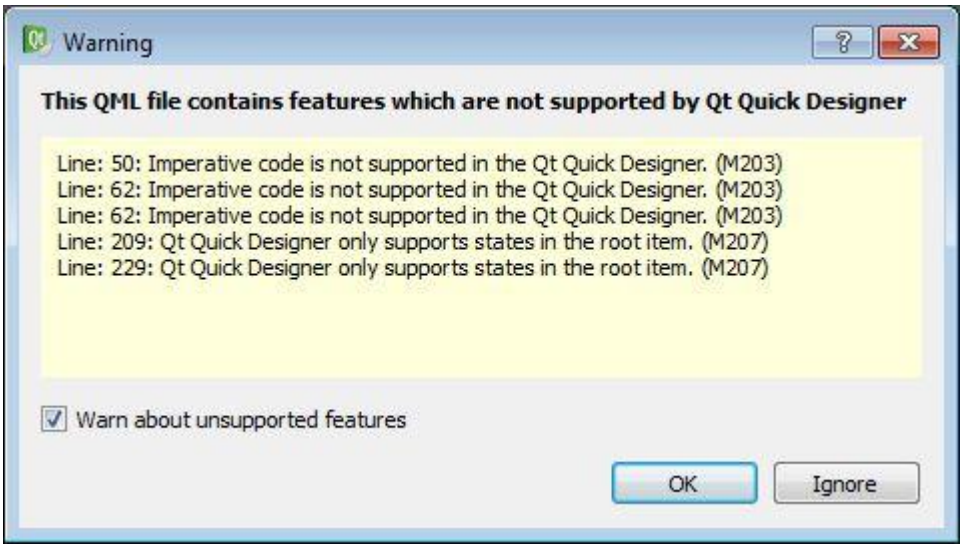
Kuvio 24. Qt Quick Designer

Designerin komponenttikirjastosta on helppoa raahata komponentteja näytön työtilaan, eli kanveesille. Komponenttikirjastossa näkyy kaikki Qt Quick:in omat komponentit ja peruselementit, sekä kaikki itse tehdyt komponentit. Kun komponentin raahaa kanveesille, sitä voi liikutella raahaamalla sen halutulle paikalla, tai ominaisuudet ikkunan kautta esimerkiksi ankkuroida sen johonkin toiseen komponenttiin. Vaihtamalla Qt Creator näkymästä välillä koodieditorin puolelle, voidaan käydä tarkistamassa, millaista koodia designeri luo.

Designerin kautta voidaan myös vaikuttaa komponenttien pinoamisjärjestykseen, merkitä käännettävät tekstielementit eri kieliversioita varten, vaihtaa vanhempi- ja lapsielementtien rooleja jne. Qt Quick Designerin kautta pystyy hyödyntämään monipuolisesti lähes kaikkia QML:n tarjoamia ominaisuuksia käyttöliittymäkehityksessä. Erityisesti prototypoinnissa designeri on loistava työkalu.

Designer on hyvä ja monipuolinen työkalu, mutta aivan kaikkia QML:n ominaisuuksia se ei tue, sillä se ei esimerkiksi ymmärrä imperatiivista koodia. Tämän lisäksi se ei tue aivan kaikkia elementtejä, kuten esimerkiksi Timer-elementtiä, eikä se tue tiloja muil-

le elementeille kuin juurielementeille. Jos qml-tiedosto sisältää ei tuettuja ominaisuuksia, Designer varoittaa asiasta käynnistyksen yhteydessä (ks. kuvio 25). Varoituksen voi ohittaa jolloin tiedosto näytetään Designerissa, mutta on hyvä muistaa, että esimerkiksi elementille JavaScriptillä tehtyjä muutoksia ei Designerissa näytetä.



Kuvio 25. Qt Quick Designer:in varoitus ei-tuetuista ominaisuuksista

6 Qt Quick prototyyppityökaluna

Jens Bache-Wiig on norjalainen Qt-kehittäjä, joka on työskennellyt Qt:n parissa jo vuodesta 2005 alkaen, jolloin Qt oli vielä Trolltech-yhtiön omistuksessa. Bache-Wiig (2011a) kertoi Qt Quick:in hyödyistä prototypoinnissa vuoden 2011 Qt DevDays -tapahtumassa. Hän korosti erityisesti sitä, kuinka paljon työtä voi mennä hukkaan kun ideoita ja ajatuksia toteutetaan ensin perinteisillä ohjelmistoilla, kuten Adobe Flash tai Photoshop, ja ongelmat huomataan vasta kun sovellusta toteutetaan kohdelaitteelle lopullisella teknologialla. (Bache-Wiig 2011a.)

Bache-Wiigin (mt.) mukaan Qt Quick -teknologian ansiosta design-suunnitteluvaiheen turha työ voidaan välttää, koska Qt Quick -prototyyppejä voidaan testata suoraan kohdelaitteissa. Tällöin jo varhaisessa vaiheessa on tiedossa kaikki lopputuotteen toteutuksen rajoitteet. Kohdelaite asettaa joitain rajoitteita sovelluksen designille, ja esimerkiksi suorituskykyyn vaikuttavat tekijät on hyvä ottaa huomioon mahdollisimman aikaisin. Qt Quick -prototyypeillä saadaan välitöntä palautetta siitä, miten sovellus toimii ja miltä se näyttää kohdelaitteessa. (Mt.)

Tämän lisäksi omien komponenttien luomisen ja uudelleen käyttämisen helppous, tekee Qt Quick -prototypoinnista tehokasta. Kun oman sovelluksen UI-komponenttikirjasto on tehty, erilaisten layouttien ja ratkaisujen kokeileminen on nopeaa. Sovelluksen kokoaminen deklaratiivisella QML-kielellä on vaivatonta ja täysin ohjelmointitaidotonkin pystyy kokoamaan sovelluksen sivuja tai komponentteja Qt Quick Designerin avulla. Lisäksi interaktioiden lisääminen sovellukseen States-ominaisuuden avulla, helpottaa ja nopeuttaa prototypointia. Tämän kaltaiset ominaisuudet voivat varmasti tiivistää graafikoiden ja kehittäjien välistä yhteistyötä.

Qt-yhteisö on kasvava ja aktiivinen yhteisö, ja esimerkiksi Qt-projektisivuston foorumeilla keskustellaan todella aktiivisesti. Aktiivinen yhteisö helpottaa teknologiaan tutustumista ja auttaa selvittämään mahdollisia ongelmia, mutta se tuo mukanaan myös muita etuja. Qt-projektisivustolla on listattuna ulkopuolisten kehittämiä lisäosia

ja kirjastoja Qt:lle (Add-ons 2013). Lisäosia on saatavilla niin avoimen lähdekoodin lisensoinneilla, kuin kaupallisillakin lisensseillä. Vaikka lisäosien/kirjastojen määrä on vielä varsin maltillinen, jos verrataan esimerkiksi HTML-kehityksessä käytettävissä oleviin JavaScript-kirjastoihin, näistäkin kirjastoista saa monenlaista apua kehitystyössä. Lisäksi suuri ja aktiivinen yhteisö yleensä takaa sen, että vastaavia kirjastoja tulee tulevaisuudessa myös lisää.

Jens Bache-Wiig julkaisi taannoin Qt-Blogissa (2010) erityisesti prototypointiin hyödyllisen skriptin Photoshopille ja GIMP:lle. Kun skriptin ajaa grafiikkaohjelmistossa, ohjelmiston valikkoon tulee uusi valinta ”Export QML”. Tämän avulla joko Photoshop, tai GIMP, osaa luoda uuden QML-tiedoston, jossa kaikki grafiikkatiedoston tasot (layer) ovat omina kuva- tai tekstielementteinä. Koska Adobe Illustratorilla pystyy tiedoston muuntamaan Photoshop-tiedostoksi, skripti toimii myös mutkan kautta vektorigrafiikoille. Valitettavasti skripti ei toimi Photoshopin uusimmilla versioilla, sillä se on testattu toimivaksi vain Photoshop CS4 ja CS5 versioissa. (Bache-Wiig 2010.)

Export QML -skripti avulla grafiikkaohjelmistossa tehdyt layout luonnostelut saa helposti ja nopeasti QML-tiedostoksi ja Qt Creatoriin käsiteltäväksi. Skripti luo jokaisesta grafiikkatiedoston juuritasosta (layer) oman elementin QML-tiedostoon ja osaa tunnistaa onko taso grafiikkaa vai tekstiä. Tekstitasot muunnetaan Text-elementeiksi ja grafiikkatasojen grafiikat viedään png-tiedostoiksi ja ne muunnetaan QML-tiedostoon Image-elementeiksi, joiden source-ominaisuus viittaa luotuun png-tiedostoon. Tällä tavalla esimerkiksi grafiikkaohjelmistossa kirjoitetut tekstit säilyvät Qt Creatorissa muokattavina ja kuvatiedostojen ominaisuuksia, kuten läpinäkyvyyttä, voi muokata Qt Creatorin kautta. Elementit sijoitellaan QML-tiedostoon x- ja y-koordinaattien avulla ja niille annetaan automaattisesti id grafiikkatiedoston tason nimen mukaan. (Bache-Wiig 2011b.)

Tämänkaltainen skripti on erinomainen, jos halutaan grafiikkaohjelmistolla toteutettuihin layoutteihin kokeilla nopeasti erilaisia toiminnallisuuksia. Jos skriptiä halutaan käyttää, se kannattaa kuitenkin huomioida jo grafiikkaa tehdessä, jotta graafikko luo

tasoja skriptiä ajatellen järkevästi ja esimerkiksi osaa nimetä tasot käyttötarkoitusta varten hyvin.

Seuraavassa taulukossa on listattuna Qt Quick -teknologian vahvuuksia ja heikkouksia prototypoinnissa (ks. taulukko 3).

Taulukko 3. QML-teknologia prototypoinnissa

+	-
Qt Quick Designer	Vähän ulkopuolisia lisäosia/kirjastoja (vrt. HTML5).
Qt Quick Components	
States-ominaisuus	
Alustariippumaton. Prototyyppiä voi testata suoraan kohdelaiteessa.	
Digia panostaa Qt-teknologiaan vahvasti.	
Aktiivinen yhteisö.	
Selkeä, paljon esimerkkejä sisältävä, dokumentaatio.	
Kehitysympäristö: Qt Creator (Qt-lisäosa asennettavissa mm. Visual Studioon). **Ohjelmointikieli**: QML/JavaScript/C++	

7 Pohdinta

Prototyyppien toteutuksessa kehittäjän kannalta Flash, HTML5 ja Qt Quick ovat teknologioina melko samanlaisia. Kaikkia yhdistää ECMAScriptiin pohjautuva ohjelmointikieli, ja HTML- ja QML-teknologioita yhdistää myös niiden deklaratiivinen sovelluksen rakenteen ja ulkoasun kuvaus. Flash ja Qt Quick tarjoavat joitain oikoreittejä prototyyppien graafisen ilmeen kehittämiseen erilaisilla graafisilla työkaluilla, mutta muuten näillä eri teknologioilla työskenneltäessä työvaiheet ovat hyvin samanlaiset. Jos kehittäjä hallitsee hyvin jonkin edellä mainitun teknologian, siirtyminen kehittämään jollain toisella mainituista teknologioista on melko vaivatonta. Jokaisella teknologialla on kuitenkin omat erityispiirteensä, jonka vuoksi jokaisella kehittäjällä on varmasti oma mielipiteensä siitä, mikä teknologia on kaikkein paras ja tehokkain mihinkin tarkoitukseen.

Prototyyppityökalua ja teknologiaa valittaessa tärkeimmät huomioitavat asiat ovat lopputuotteen toteutusteknologia sekä kohdealusta. Jos prototyyppi on mahdollista toteuttaa samalla teknologialla kuin lopputuote, se on kannattavaa ainakin näistä kolmesta teknologiasta puhuttaessa. Kun prototyypit toteutetaan samalla teknologialla kuin lopputuote, tiedetään jo prototypointi vaiheessa teknologian asettamat rajoitteet ja vaatimukset sovellukselle. Lisäksi päästään mahdollisesti testaamaan prototyyppiä kohdelaitteessa, jolloin voidaan jo prototyypissä huomioida esimerkiksi suorituskykyyn vaikuttavia asioita. Tällä tavalla prototyyppi on myös mahdollista kehittää aina lopputuotteeksi asti, kunhan tämä aikomus pidetään mielessä heti projektin alusta asti.

Opinnäytetyössä tutustuttiin erityisesti Qt-teknologian tarjoamiin mahdollisuuksiin prototypoinnissa. Qt osoittautui todella alustariippumattomaksi teknologiaksi ja koska sillä toteutetut sovellukset ajetaan kohdelaitteessa natiivisti, se on myös hyvin suorituskykyinen. Käyttöliittymäkerros on Qt Quick -teknologian avulla mahdollista toteuttaa moderneilla työkaluilla ja teknologioilla, mutta ohjelmistologiikka voidaan

toteuttaa erikseen vakaalla C++ teknologialla. Tämä on erityisen hyödyllistä kun kehitetään natiiveja sovelluksia sulautetuille järjestelmille.

Qt on erittäin monipuolinen teknologia, ja Qt Quick:in vuoksi se on myös erinomainen työkalu ketterään prototypointiin. UI/UX-suunnittelijat ja ohjelmoijat voivat työskennellä samoilla työkaluilla jo projektin ensi askeleista lähtien. Opinnäytetyön teknologiatutkimuksen perusteella voidaan todeta, että Qt Quick on erittäin monipuolinen, helppokäyttöinen ja ketterä työkalu prototypointiin. Lisäksi Qt Quick -prototyypin jatkokehittäminen lopputuotteeksi eri alustoille onnistuu helposti, sillä se on mahdollista tarvittaessa yhdistää C++-taustaan. Tämän vuoksi toimeksiantajan ei kannata korkean tarkkuuden prototyyppikehityksessä käyttää muita teknologioita, kuin Qt Quick -teknologiaa. Qt Quick soveltuu erittäin hyvin myös karkeiden matalan tarkkuustason pois-heitto-prototyyppien toteutukseen, mutta tällaiset prototyypit on järkevintä toteuttaa kehittäjän kannalta helpoimmalla, tutuimmalla ja nopeimmalla teknologialla, sillä niitä ei ole tarkoituskaan kehittää lopputuotteeksi.

Lähteet

About us. 2013. Qt:n lyhyt historia Digian www-sivustolla. Viitattu 3.12.2013. http://qt.digia.com/About-us/

Add-ons. 2013. Listaus ulkopuolisista lisäosista ja kirjastoista Qt-projektin www-sivustolla. Viitattu 28.12.2013. http://qt-project.org/wiki/Category:Add-ons

Adobe Creative Cloud. 2013. Usein Kysytyt Kysymykset. Viitattu 28.11.2013. http://www.adobe.com/fi/products/creativecloud/faq.html

Adobe Creative Cloud / Jäsenyysohjelmat. 2013. Työryhmät ja yritykset hinnoittelu. Viitattu 28.11.2013. http://www.adobe.com/fi/products/creativecloud/buying-guide-teams-enterprise.html

Ambler, S. 2001. The object primer: The application developer's guide to object orientation and the UML, Second edition. Viitattu 26.11.2013. Http://www.jamk.fi/kirjasto, Nelli-portaali, books24x7.

Arnowitz, J., Arent, M. & Berger, N. 2006. Effective prototyping for software makers. Viitattu 26.11.2013. Http://www.jamk.fi/kirjasto, Nelli-portaali, Ebrary.

Bache-Wiig, J. 2010. Julkaisu Qt Blogissa. Viitattu 28.12.2013. http://blog.qt.digia.com/blog/2010/10/19/exporting-qml-from-photoshop-and-gimp/

Bache-Wiig, J. 2011a. Videotallenne Qt DevDays -seminaarista QtStudios:in YouTube-kanavalla. Viitattu 17.12.2013. http://www.youtube.com/watch?v=k1pnEhyF1pg

Bache-Wiig, J. 2011b. Opastusvideo YouTube-videopalvelussa. Viitattu 30.12.2013. http://www.youtube.com/watch?v=9BRJ1qJkjZM

Cerejo, L. 2010. Design better and faster with rapid prototyping. Artikkeli smashing-magazine:n sivustolla. Viitattu 26.11.2013. http://www.smashingmagazine.com/2010/06/16/design-better-faster-with-rapid-prototyping/

Creating Projects. 2013. Qt dokumentaatio. Viitattu 10.12.2013. http://qt-project.org/doc/qtcreator-2.8/creator-project-creating.html

Digia julkaisi Qt 5.1. 2013. Digian www-sivuilla 3.7.2013 julkaistu uutinen. Viitattu 2.12.2013. http://www.digia.com/fi/Yritys/Uutiset/Digia-releases-Qt-51/

Digia ostaa koko Qt-kehitysympäristön Nokialta. 2012. Digian www-sivuilla 9.8.2012 julkaistu uutinen. Viitattu 3.12.2013. http://www.digia.com/fi/Yritys/Lehdisto/2012/Digia-ostaa-koko-Qt-kehitysympariston-Nokialta/

Digia ostaa Qt-ohjelmistojen kaupallisen lisensointi- ja palveluliiketoiminnan Nokialta. 2011. Digian www-sivuilla 7.3.2011 julkaistu uutinen. Viitattu 3.12.2013. https://digia.com/fi/Yritys/Lehdisto/2011/Digia-ostaa-Qt-ohjelmistojen-kaupallisen-lisensointi--ja-palveluliiketoiminnan-Nokialta/

Flash Professional CC / Features. 2013. Tuote-esittely Adoben sivustolla. Viitattu 28.11.2013. http://www.adobe.com/fi/products/flash/features.html

Grossman, G. & Huang, E. 2006. ActionScript 3.0 overview. Viitattu 28.11.2013. http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html

Hartson, R. & Pyla, S. 2012. The UX Book: Process and guidelines for ensuring a quality user experience. Viitattu 26.11.2013. Http://www.jamk.fi/kirjasto, Nelli-portaali, books24x7.

Hix, D. & Hartson, R. 1993. Developing user interfaces: Ensuring usability through product & process. Viitattu 26.11.2013. Http://www.jamk.fi/kirjasto, Nelli-portaali, books24x7.

IDE Overview. 2013. Qt Creator esittely Qt projektin sivustolla. Viitattu 10.12.2013. http://qt-project.org/doc/qtcreator-2.8/creator-overview.html

Item positioners. 2013. Qt 5.1 dokumentaatio. Viitattu 5.12.2013. http://qt-project.org/doc/qt-5.1/qtquick/qtquick-positioning-layouts.html#flow

JavaScript Expressions in QML Documents. 2013. Qt 5.0 dokumentaatio. Viitattu 5.12.2013. http://qt-project.org/doc/qt-5.0/qtqml/qtqml-javascript-expressions.html

jQuery user interface. 2013. Www-sivusto. Viitattu 29.11.2013. http://jqueryui.com/

Kolehmainen, A. 2012. Html5 on raakaa työtä. Tietoviikko 13, 7.

Larsen, R. 2013. Beginning HTML and CSS. Viitattu 29.11.2013. Http://www.jamk.fi/kirjasto, Nelli-portaali, books24x7.

Pollock, J. 2013. JavaScript: A Beginners Guide, Fourth Edition. Viitattu 29.11.2013. Http://www.jamk.fi/kirjasto, Nelli-portaali, books24x7.

Qt 5.1 available. 2013. Qt 5.1:n esittely Qt Digian www-sivustolla. Viitattu 3.12.2013. http://qt.digia.com/Product/Whats-New/Qt-51/

Qt 5.2 Beta Available. 2013. Qt 5.2 Betan:n esittely Qt Digian www-sivustolla. Viitattu 3.12.2013. http://blog.qt.digia.com/blog/2013/10/23/qt-5-2-beta-available/

Qt Creator IDE and tools. 2013. Qt Creatorin esittely Digian Qt-sivustolla. Viitattu 10.12.2013. http://qt.digia.com/Product/Qt-Core-Features-Functions/Developer-Tools/

Qt Quick. 2013. Teknologiaesittely Digian sivustolla. Viitattu 3.12.2013. https://qt.digia.com/Product/Qt-Core-Features-Functions/qt-quick/

Qt Quick Controls Styles. 2013. Qt 5.1 dokumentaatio. Viitattu 11.12.2013. http://qt-project.org/doc/qt-5.1/qtquickcontrolsstyles/qtquickcontrolsstyles-index.html

Qt Quick Layouts. 2013. Qt 5.1 dokumentaatio. Viitattu 3.12.2013. http://qt-project.org/doc/qt-5.1/qtquicklayouts/qtquicklayouts-index.html

Sharp, H., Rogers Y. & Preece, J. 2007. Interaction design: Beyond human-computer interaction, Second edition. Viitattu 27.11.2013. Http://www.jamk.fi/kirjasto, Nelli-portaali, books24x7.

Signal and handler event system. 2013. Qt 5.1 dokumentaatio. Viitattu 10.12.2013. http://qt-project.org/doc/qt-5.1/qtqml/qtqml-syntax-signals.html

Tietoa Valtrasta. 2013. Valtra Oy:n www-sivusto. Viitattu 2.12.2013. http://www.valtra.fi/company/128.asp

Use Case - Positioners and Layouts in QML. 2013. Qt 5.1 dokumentaatio. Viitattu 3.12.2013. http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-usecase-layouts.html

Use Case – Visual Elements in QML. 2013. Qt 5.1 dokumentaatio. Viitattu 5.12.2013. http://qt-project.org/doc/qt-5.1/qtdoc/qtquick-usecase-visual.html

Uusi mobiiliympäristöön suunnattu Qt-tuote tehostaa sovelluskehitystä. 2013. Digian www-sivuilla 8.10.2013 julkaistu uutinen. Viitattu 2.12.2013. http://www.digia.com/fi/Yritys/Uutiset/Uusi-mobiiliymparistoon-suunnattu-Qt-tuote-tehostaa-sovelluskehitysta/

What's new in Qt 5.1. 2013. Qt 5.1:n uusien ominaisuuksien esittely Qt-projektin www-sivustolla. Viitattu 3.12.2013. http://qt-project.org/doc/qt-5.1/qtdoc/whatsnew51.html