HAAGA-HELIA
ammattikorkeakoulu

Kielenkääntäjän sanastotyökalu

Markus Becks

Opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

2013

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

31.12.2013

Tietojenkäsittelyn koulutusohjelma

Tekijä tai tekijät Markus Becks	**Ryhmätunnus tai aloitusvuosi** 2009
Raportin nimi Kielenkääntäjän sanastotyökalu	**Sivu- ja liitesivumäärä** 39+7
Opettajat tai ohjaajat Markku Kuitunen	

Tämän työn tarkoituksena oli tuottaa käännöspalvelu Databexit Oy:lle kielenkääntäjän sanastotyökalu, jonka avulla kielenkääntäjä pystyy luomaan omia käännöstyökohtaisia tai toimialakohtaisia sanastojaan. Työ aloitettiin kesällä 2013 ja saatettiin loppuun jouluna 2013.

Kielenkääntäjä tarvitsee monissa käännöstöissään erikoissanastoja, jotka usein syntyvät käännöstyön aikana. Näitä sanastoja hyödynnetään ja täydennetään sekä meneillään olevassa käännöstyössä että myöhemmin, kun tehtäväksi tulee käännöksiä samasta aiheesta. Näin kääntäjä voi käyttää termejä johdonmukaisesti ja yhtenevästi niin meneillään olevassa käännöstyössä kuin myös myöhemmissä saman aiheen käännöksissä.

Käyttäjän tuli voida hakea, lisätä, muokata ja poistaa sanoja. Jokaisella sanaparilla piti olla oma kommenttikenttänsä, toimialansa ja päivämääränsä, joita käyttäjä voi hyödyntää erilaisissa monipuolisissa hauissa ja kategorisoinnissa. Sovelluksen tuli olla helppokäyttöinen, ja sen piti toimia nopeasti.

Sovellus toteutettiin täysin ilmaisilla välineillä ja tekniikoilla: ohjelmointiympäristö oli Visual Studio Express ja ohjelmointikieli C#. Sovelluksessa hyödynnettiin .NET-ympäristöä, ja sovellus sisältää SQLite-tietokannan.

Sovelluksen toimintaa testattiin Databexit Oy:n omistamalla 117 000 sanaparin sanakirjalla, joka tuotiin sovellukseen ohjelmallisesti. Iso testisanasto antoi arvokasta teknistä tietoa erityisesti haun nopeudesta, minkä perusteella sovelluksen tehokkuutta voitiin kehittää. Sovelluksen käytettävyys testattiin projektin aikana ominaisuus kerrallaan, ja lopuksi suoritettiin yhtenäinen, jokaisen ominaisuuden kattava käytettävyystestaus. Toimeksiantaja osallistui käytettävyystestaukseen ja antoi palautetta toteutetuista ominaisuuksista.

Asiasanat
Käytettävyys, .NET, C#, Windows Forms, SQLite

HAAGA-HELIA
University of Applied Sciences

Abstract

31.12.2013

Degree Programme in Business Information Technology

Authors Markus Becks	**Group or year of entry** 2009
The title of thesis Translator's glossary tool	**Number of report pages and attachment pages** 39+7
Advisor(s) Markku Kuitunen	

Translators frequently need special glossaries in their work. These glossaries are usually compiled during the translation process. The specialized glossaries are utilized and complemented while the translation work is in progress, as well as later, during various translation tasks related to the same special field or sector. Consequently, the translator will be able to use the terms in a consistent and uniform manner.

The purpose of this thesis was to create a professional translator's glossary tool for Databexit Oy. By using this tool, translators would be able to create their own field-specific or project-based glossaries on their various translation projects. The thesis was started in the summer of 2013 and completed at the end of the year 2013.

The application was to enable the translator to search, add, edit and delete words. In addition, each word pair had to have its own field for comments, a special field for a particular project and the date. The translator can use these fields in various searching modes and while categorizing the data. Moreover, the application was to be easy and quick to use.

The application was developed using a set of tools and technologies that are completely free of charge. Visual Studio Express was chosen as the interactive development environment and C# as the programming language. .NET Framework was utilized to develop the application and it contains a SQLite database.

The functional testing of the application was carried out with a dictionary of 117 000 word pairs owned by Databexit Oy. The dictionary was imported into the application by using a program. The extensive test glossary yielded valuable technical information, especially on search speed. This information was used to enhance the search efficiency. After developing a single new feature for the application, a usability test took place. At the end of the development process, a more comprehensive usability test was conducted. The client participated in the testing and gave feedback on the features implemented in the application.

Key words
Usability, .NET, C#, Windows Forms, SQLite

Sisällys

Sanasto

.NET Framework: Microsoftin kehitysalusta, jonka avulla voidaan rakentaa sovelluksia Windows-ympäristöön. (Microsoft Developer Network c.)

API: Ohjelmointirajapinta (engl. Application programming interface)

Windows Forms: .NET Frameworkin sisältämä graafinen ohjelmointitekniikka. (Microsoft Developer Network g.)

SQLite: Ilmainen ohjelmistokirjasto, joka sisältää palvelimettoman, omavaraisen tietokantamoottorin. (SQLite a.)

C#: Microsoftin kehittämä ohjelmointikieli .NET Frameworkille. (ECMA International 2006, 21.)

IDE: Ohjelmointiympäristö (engl. Integrated development environment)

Erikoissanasto: Kielenkääntäjän hyödyntämä kokoelma sanoja, jotka ovat tyypillisiä tietylle aihepiirille tai alalle.

Sanapari: Käännettävän sanan ja käännöksen muodostama sanojen pari.

Alkuteksti: Teksti, joka kielenkääntäjän on tarkoitus kääntää.

XML: (Extensible Markup Language) on joukko tekstiformaattien suunnittelun sääntöjä, jonka avulla voi jäsentää tietoa. (W3C 2001.)

Käytettävyys: Koostuu opittavuudesta, tehokkuudesta, muistettavuudesta, virheiden määrästä ja käyttäjän tyytyväisyydestä. (Nielsen 1993, 26.)

SQL-injektio: Joukko varmistamatonta käyttäjän antamaa syötettä, käytännössä SQL-koodia, jonka uskotellaan sovellukselle toimivan tarkoitetulla tavalla. (Friedl 2007.)

1 Johdanto

Opinnäytetyön tavoitteena oli tuottaa Databexit Oy:lle kielenkääntäjän erikoissanastotyökalu, jonka avulla kääntäjä pystyy helposti kategorisoimaan, etsimään, lisäämään ja muokkaamaan tietyn toimialan tai työn sanoja. Käytännössä työkalu korvaa kielenkääntäjän käyttämät lukuisat toimialakohtaiset Word-tiedostot yhdellä ohjelmalla ja tarjoaa samalla toimiala- tai asiakaskohtaisen sanastosovelluksen. Työkalu ei nykyisessä muodossaan tule täysin korvaamaan kielenkääntäjän käyttämiä sanakirjoja tai sanastoja vaan täydentämään niitä.

Kielenkääntäjä tarvitsee käännöstyössään kuhunkin työhön liittyvää erikoissanastoa. Sanasto syntyy kääntäjän tehdessä käännöstä, ja tätä sanastoa hyödynnetään ja täydennetään sekä meneillään olevassa käännöstyössä että myöhemmin, kun tehtäväksi tulee käännöksiä samasta aiheesta. Näin kääntäjä voi käyttää termejä johdonmukaisesti ja yhdenmukaisesti niin meneillään olevassa käännöstyössä kuin myös myöhemmissä saman alan käännöksissä.

Sovelluksen toteutus- ja ylläpitokustannukset haluttiin pitää alhaisina, mikä vaikutti käytettyjen työkalujen ja tekniikoiden valintaan suuresti. Kaikki sanastotyökalun toteutuksessa käytetyt sovellukset olivat ilmaisia. Sovellus ohjelmoitiin C#-ohjelmointikielellä Microsoftin .NET-ympäristöön Windows Forms API -tekniikalla. Ohjelma toteutettiin Microsoft Visual Studio 2012 Express - ja Microsoft Visual Studio 2013 Express -versioilla. Työkalu on tarkoitettu työpöytäkäyttöön, ja se käyttää luomaansa SQLite-tietokantaa. Sovellus ei ole yhteydessä verkkoon millään tavalla.

Sovelluksen kannalta oleellista oli käytettävyys. Sovelluksen tuli toimia käyttäjälle nopeasti ja vaivattomasti. Tähän pyrittiin kiinnittämään huomiota teknisessä toteutuksessa ja käyttöliittymän suunnittelussa.

1.1 Toimeksiantaja

Databexit Oy on vuonna 1994 perustettu perheyritys, joka tarjosi IT-alan konsultointia ja käännöspalveluita. Nykyään yritys keskittyy enimmäkseen englanti – suomi – englanti -käännöspalveluihin.

Sovellus tulee vain toimeksiantajan omaan käyttöön. Sovellusta ei siis ainakaan kehityksen aikana ollut tarkoitettu laajempaan käyttöön, mikä näkyi ominaisuuksien toteutuksessa varsin yksilöityinä ratkaisuina. Lähtökohta sovelluksen kehitykseen oli se, että sovellus suunnitellaan suoraan toimeksiantajan haluamien vaatimusten perusteella. Toimeksiantaja määritteli sovelluksen vaatimukset itse, osallistui sovelluksen käytettävyystestaukseen ja antoi palautetta toteutetuista ominaisuuksista projektin aikana.

1.2 Työn tavoite

Työn tavoitteena oli toteuttaa vaatimusten mukainen kielenkääntäjän sanastotyökalu. Sovelluksen yksi tärkeimmistä ominaisuuksista on se, että kääntäjä voi koota alkutekstin perusteella toimiala- tai asiakaskohtaisia sanastoja, jotta hän voi käyttää termejä työn aikana ja myöhemmissä saman asiakkaan tai aiheen töissä johdonmukaisesti uudestaan. Kääntäjällä on usein Word-tiedosto, jonka päälle hän tekee käännöksen. Toisinaan alkuteksti on vain paperidokumentti, jolloin alkutekstin sana pitää kirjoittaa sanastoon eikä sitä voida siirtää sinne kopioi-liimaa-komennolla niin kuin silloin, jos alkuteksti on Word-tiedosto.

Aikaisemmin johdannossa mainittujen perustoiminnallisuuksien lisäksi vaatimukset liittyivät lähinnä sovelluksen käyttönopeuteen ja helppouteen. Sanahakujen tuli toimia nopeasti, ja sovelluksen käyttöliittymää tuli olla helppo käyttää. Yhtenä toivomuksena oli sovelluksen hiirtä vaativien toiminnallisuuksien minimointi. Tämän perusteella sovelluksen käyttöliittymä toteutettiin niin, että sitä voidaan haluttaessa käyttää lähes pelkästään pikanäppäinten ja näppäimistön avulla.

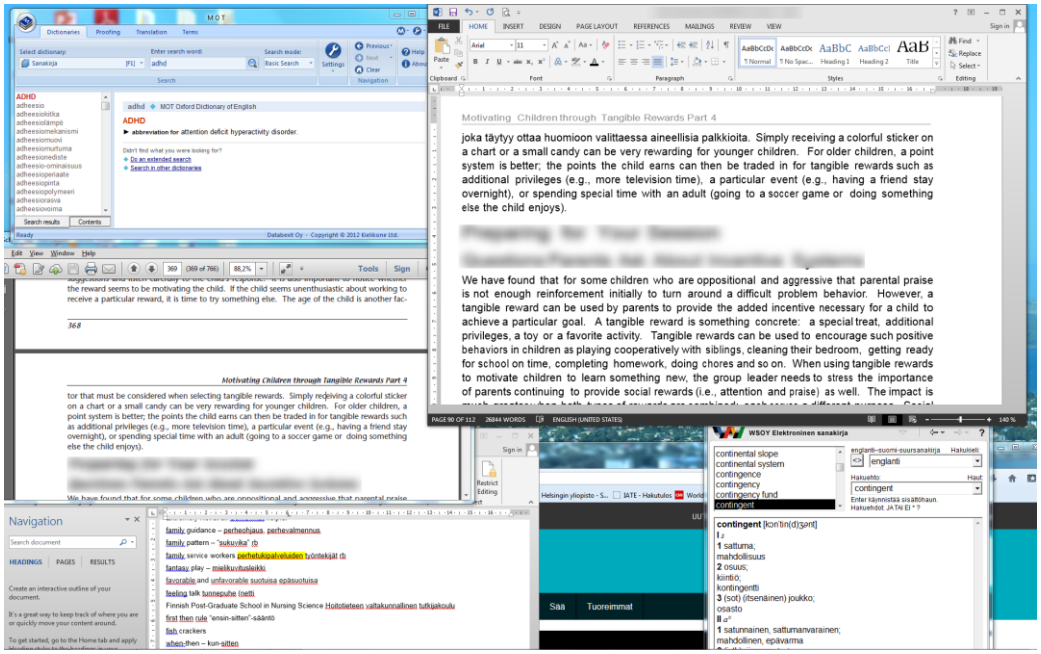
2 Tietoperusta

Tässä luvussa käydään läpi projektin aikana käytetyt työkalut, teknologiat ja oleellinen teoria sekä kerrotaan, miksi valittuihin ratkaisuihin päädyttiin.

2.1 Sovelluksen käyttöympäristö

Projektin kannalta oli tärkeää tutustua sovelluksen tulevaan käyttöympäristöön ja käyttäjän nykyisiin työskentelytapoihin. Jotta sovellus olisi hyödyllinen, piti sen suunnittelussa ottaa edellä mainitut asiat huomioon ja rakentaa sovellus niiden perusteella.

Ohjelman tulevalla käyttäjällä on työpöydällään auki lukuisia eri ohjelmia ja ikkunoita, jotka on sijoiteltu ruudulle siten, että hän näkee tietyt osat haluamistaan ikkunoista. (Kuva 1) Ruudulla ohjelma ottaa yhden Word-dokumentin paikan. Tämän takia sovelluksen ikkunat pyrittiin suunnittelemaan niin, että niitä olisi mahdollista käyttää myös pienessä tilassa.



Kuva 1. Työpöydältä otettu kuvakaappaus, joka havainnollistaa sovellusten ja ikkunoiden tyypillistä sijoittumista tietokoneen ruudulle. Ruudulla on auki alkuteksti pdf-muodossa, Word-dokumentti, johon käännös tehdään, internet-selain, kaksi

sanakirjaa ja toinen Word-dokumentti, johon kääntäjä on laatimassa omaa käännöskohtaista sanastoaan. Sovellus tulee toisen Word-dokumentin paikalle ruudun vasempaan alakulmaan.

Käännöstyöt vaihtelevat suuresti satojen sivujen pituisista manuaaleista yhden sivun töihin, joten tyypillisen käännöstyön pituutta tai rakennetta on erittäin vaikea määrittää. Joissakin töissä tarkkojen, toimialakohtaisten sanojen määrä suhteessa sanojen kokonaismäärään on hyvinkin suuri, kun taas toisissa töissä toimialakohtaista sanastoa on vähemmän.

Kääntäjällä on käytössään lukuisia sanastoja, joista jotkut ovat kaupallisia, toiset taas ilmaisia. Sanastoista on hyötyä nimenomaan sanojen ja termien haussa, mutta helppoa ja yksinkertaista mahdollisuutta omien sanastojen luomiseen ei niissä käyttäjän mielestä ole. Tästä johtuen käyttäjä usein rakentaa työ- tai toimialakohtaiset sanastot erillisille Word-dokumenteille. Erillisen, uuden sovelluksen rakentamista pidettiin tarpeellisena, koska Word ei tarjoa tarpeeksi ominaisuuksia systemaattiseen sanaston luomiseen ja järjestämiseen. Esimerkiksi yksittäisiä sanapareja ei voida hakea järkevästi päivämäärän perusteella, eikä sanastojen ja käännösten karsinointiin ole käyttäjän mielestä käytännöllisiä vaihtoehtoja. Excelillä olisi luultavasti voitu toteuttaa palvelu, joka olisi vastannut suurinta osaa vaatimuksista, mutta käyttäjä ei pitänyt tästä ideasta, koska Excel on hänelle suhteellisen vieras ja sen käyttöliittymä hankala. Lisäksi käyttäjäystävällisen käyttöliittymän toteuttaminen Excel-ympäristöön olisi ollut melko vaikeaa.

Sovellus oli tarkoitettu yhden henkilön kotitoimistokäyttöön, mistä johtuen tuntui luontevalta toteuttaa ohjelma työpöytäsovellukseksi. Toinen vaikuttava tekijä päätökseen rakentaa työpöytäsovellus oli oppiminen; ammattikorkeakoulun opintojen aikana on keskitytty ainoastaan verkkosovelluksiin. Työpöytäsovelluksen tekeminen oli siis jo sinällään hyödyllinen ja kiinnostava oppimiskokemus.

Toinen vaihtoehto olisi ollut rakentaa verkkosovellus. Verkkosovelluksen etu, mutta myös haitta, on sen saatavuus. Verkkosovellukseen pääsee käsiksi mistä vain, mutta se vaatii internet-yhteyden. Mikäli käännöstyökalu olisi ollut laajemmassa käytössä, olisi

verkkosovellus voinut olla järkevä vaihtoehto. Verkkosovelluksen muita hyötyjä on sen tukemisen ja ylläpidon helppous verrattuna työpöytäsovellukseen. (Microsoft Developer Network b.) Verkkoon toteutettu sovellus tuo automaattisesti mukanaan myös tietoturvariskin. Huomion kiinnittäminen tietoturvaan ei ollut tässä tapauksessa olennaista, koska sovellus kehitettiin yhden henkilön yksityiskäyttöön. Lisäksi tietoturvaan keskittyminen olisi vienyt aikaa muulta toteutukselta. Toteuttamalla työpöytäsovellus voitiin tietoturvakysymys ohittaa käytännössä kokonaan, koska sovellus ei ole yhteydessä verkkoon ja tietokoneeseen pääsee käsiksi vain yksi henkilö.

2.2 Käytettävyys

Käytettävyyden uranuurtaja Jakob Nielsen näkee käytettävyyden koostuvan viidestä ominaisuudesta: opittavuudesta, tehokkuudesta, muistettavuudesta, virheiden määrästä ja tyytyväisyydestä. (Nielsen 1993, 26.)

Opittavuus on yksi tärkeimmistä käytettävyyden ominaisuuksista, koska useimpien järjestelmien täytyy olla helposti opittavissa ja yleensä käyttäjien ensimmäinen kosketus järjestelmään liittyy sen opetteluun. Mitä nopeammin käyttäjä oppii järjestelmän, sitä nopeammin hän voi aloittaa työskentelyn. Toisaalta järjestelmän helppous ei suoraan kuvasta järjestelmän potentiaalista tehokkuutta. Taitavat käyttäjät voivat opetella käyttämään monimutkaista käyttöliittymää, vaikka heillä siihen meneekin enemmän aikaa kuin aloittelevilla käyttäjillä helpon käyttöliittymän oppimiseen. Kun taitava käyttäjä osaa hyödyntää kaikkia monimutkaisen järjestelmän ominaisuuksia, päästään parempiin tuloksiin tehokkuuden kannalta kuin aloittelevien käyttäjien helposti opittavilla järjestelmillä. (Nielsen 1993, 26-28.)

Tehokkuus on tärkeää, koska se vaikuttaa suoraan tuottavuuteen. Tehokkuudella tarkoitetaan sitä, että taitava käyttäjä osaa käyttää lähes kaikkia monimutkaisen järjestelmän ominaisuuksia hyödykseen. (Nielsen 1993, 30.)

Muistettavuus on tärkeää erityisesti satunnaisille käyttäjille, jotka käyttävät ohjelmaa silloin tällöin. Heidän tulisi pystyä muistamaan, miten ohjelmaa käytetään pitkienkin käyttötaukojen jälkeen. Satunnaisesti käytettävät ohjelmat ovat tyypillisesti sellaisia,

jotka ovat hyödyllisiä, mutta joita ei tarvitse kuin silloin tällöin. Näitä ovat esimerkiksi erilaiset raportointiohjelmat. (Nielsen 1993, 31.)

Käyttäjän tekemien virheiden määrä ohjelman käytön aikana tulisi olla pieni, ja virheistä pitäisi pystyä toipumaan. On myös tärkeää, ettei katastrofaalisia virheitä tapahdu. Pienen virheen voidaan ajatella olevan käyttäjän suorittama mikä tahansa toimenpide, joka ei auta saavuttamaan haluttua päämäärää. Tällaisilla virheillä ei välttämättä ole muuta kuin hieman hidastava vaikutus käyttöön, jos käyttäjä pystyy heti virheen jälkeen kuitenkin saavuttamaan päämääränsä. Virheet, joita käyttäjä tekee tietämättään saaden siten aikaan esimerkiksi viallisen tuotteen, ovat ns. katastrofaalisia virheitä. Katastrofaalisiin virheisiin kuuluvat myös virheet, joiden seurauksena käyttäjä menettää tekemänsä työn tulokset. Katastrofaalisista virheistä on vaikea toipua, ja siksi niiden määrän minimoimiseen tulisi kiinnittää erityistä huomiota. (Nielsen 1993, 32-33.)

Tyytyväisyys on käyttäjälle tärkeää. Kun ohjelmaa on mukava käyttää, pitää käyttäjä ohjelmasta enemmän. Tyytyväisyys on subjektiivista, ja se koskee erityisesti ohjelmia tai järjestelmiä, jotka eivät liity työntekoon. Esimerkiksi pelien kannalta on ehkä tärkeämpää, kuinka viihdyttävä pelikokemus on, kuin kuinka nopeasti asiat saavutetaan. Käyttäjä haluaa pitää hauskaa hyvän tovin. (Nielsen 1993, 33.)

Käytettävyysprosessissa ensimmäinen askel on tuntea käyttäjä. Yksilölliset ominaisuudet ja tehtävien vaihtelevuus vaikuttavat käytettävyyteen suuresti, joten niiden tutkiminen on tärkeää. Käyttäjistä puhuttaessa on hyvä pitää mielessä, että ohjelmaa saatetaan tulla hyödyntämään myös asennuksen ja ylläpidon yhteydessä. Järjestelmänvalvojat ja tukihenkilöstö kuuluvat myös käytön piiriin, joten on luonnollista, että kaikki edellä mainitut ryhmät voidaan ajatella ohjelman käyttäjiksi. (Nielsen 1993, 73.)

Usein käyttäjien lähtötaso ja osaaminen on helppo tietää, esimerkiksi silloin, kun tuote tulee käyttöön tietyn yrityksen tietylle osastolle. Kun käyttäjien koulutustaso, työkokemus, ikä, tietotekniset taidot ja niin edelleen ovat tiedossa, käyttöliittymä on helpompi suunnitella, koska käyttäjien oppimisrajoitteita voidaan ainakin osittain arvioida. Käyttöliittymää suunniteltaessa on hyödyllistä olla perillä siitä, aiotaanko

ohjelman käyttäjiä kouluttaa millään tavalla sen käyttämiseen ja paljonko heillä on aikaa sen oppimiseen. Mikäli koulutusta ei juuri ole, tulee käyttöliittymän olla suhteellisen yksinkertainen. (Nielsen 1993, 74-75.)

Käyttöympäristö ja sosiaalinen konteksti tulee myös huomioida käyttöliittymää suunniteltaessa. Avokonttoriin tarkoitettuun tuotteeseen ei esimerkiksi kannata suunnitella voimakasäänisiä varoitusääniä sisältäviä käyttöliittymiä.

2.2.1 Käytettävyyden mittaaminen

Vaikka kaikkea käytettävyyttä ei voida suoraviivaisesti mitata (esimerkiksi subjektiivinen tyytyväisyys), voidaan käytettävyyttä arvioitaessa keskittyä esimerkiksi seuraaviin mitattaviin asioihin: (Nielsen 1993, 193-194.)

- kuinka kauan tehtävän suorittaminen kestää
- kuinka monta tehtävää tai kuinka suuri osa isosta tehtäväkokonaisuudesta voidaan suorittaa tietyssä ajassa
- onnistuneiden ja virheellisten toimenpiteiden suhteeseen
- virheistä toipumiseen käytettyyn aikaan
- virheiden määrään
- ominaisuuksien määrään, joita käyttäjä hyödynsi testin aikana
- ominaisuuksien määrään, joita käyttäjä ei hyödyntänyt testin aikana
- kuinka usein käyttäjä turvautui manuaaliin tai muuhun apuun, ja kauanko aikaa käytettiin apuominaisuuksien tutkimiseen
- kuinka usein apuominaisuudet ratkaisivat käyttäjän ongelman.

2.2.2 Käytettävyyden testaaminen

Käytettävyyden testaaminen oikeilla käyttäjillä kuuluu käytettävyystestauksen perusmenetelmiin. Sen avulla saadaan suoraa tietoa siitä, miten henkilö käyttää tietokonetta, ja mitä ongelmia tulee esille konkreetista käyttöliittymää testattaessa.

Käytettävyystestauksessa on useita sudenkuoppia. Erityisesti huomiota tulee kiinnittää käytettävyystestauksen luotettavuuteen ja oikeellisuuteen. (Nielsen 1993, 165.)

Luotettavuus (engl. reliability) on käytettävyystestauksessa ongelma käyttäjien yksilöllisyyden takia. Ei ole harvinaista, että ohjelman paras käyttäjä suorittaa jonkun tehtävän kymmenen kertaa nopeammin kuin ohjelman hitain käyttäjä tai että parhaat 25% ohjelman käyttäjistä ovat yleensä noin kaksi kertaa nopeampia kuin hitaimmat 25% käyttäjistä. Tämän takia johtopäätösten tekeminen yksittäisten käyttäjien perusteella voi olla ongelmallista. Toisaalta edes jonkinlainen testidata on parempi, kuin ettei testidataa olisi ollenkaan. Tilastollisia menetelmiä voidaan hyödyntää luotettavuuden tukemisessa. Esimerkiksi kahden käyttöliittymän käytettävyystestaustulosten luotettavuutta voidaan verrata toisiinsa luottamusvälin avulla. (Nielsen 1993, 166.)

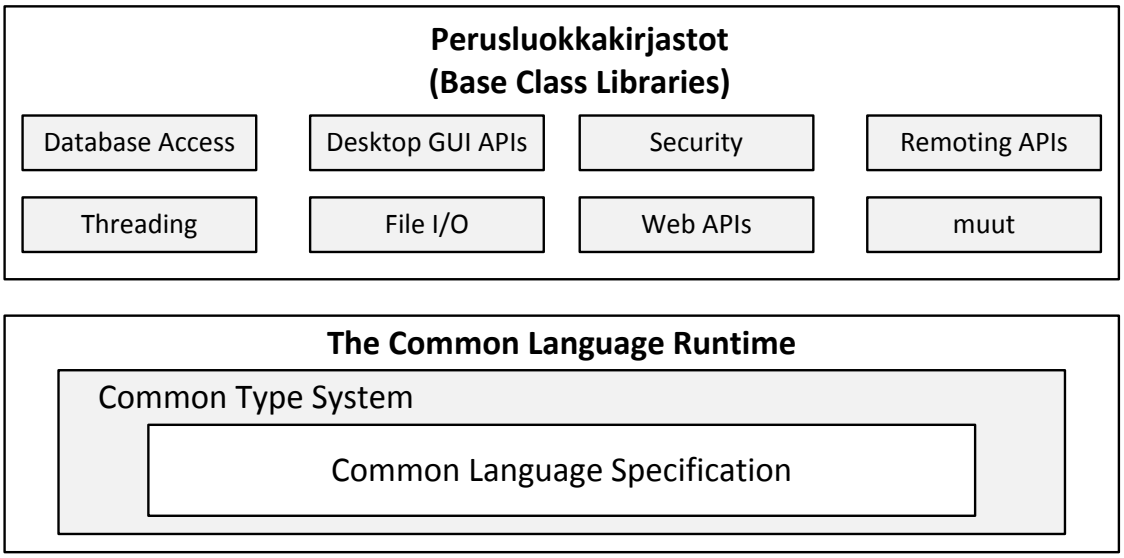
Oikeellisuudella (engl. validity) tarkoitetaan, kertooko tai mittaako käytettävyystesti todella sitä, mitä halutaan mitata. Kun luotettavuutta voidaan käsitellä tilastollisilla menetelmillä, oikeellisuus taas vaatii testitulosten tulkitsijalta testimenetelmän ymmärtämistä ja maalaisjärkeä. Tyypillinen oikeellisuuden pulma on väärien käyttäjien tai väärien tehtävien valitseminen käytettävyystestiin. Esimerkiksi johdon tietojärjestelmää voitaisiin hyvin testata liiketalouden opiskelijoilla, mutta testistä saatavat tulokset olisivat luultavasti erilaisia, mikäli testauksessa olisi käytetty opiskelijoiden sijasta johtajia. Kuitenkin liiketalouden opiskelijoiden käyttö kyseisessä testissä on perusteltavissa, koska heistä todennäköisesti tulee jonkinlaisia johtajia. Testitulos on tässä tapauksessa varmasti pätevämpi, kuin jos testiryhmäksi olisi otettu liiketalouden opiskelijoiden sijasta esimerkiksi ryhmä lääketieteen opiskelijoita. (Nielsen 1993, 169.)

2.3 .NET Framework

Ohjelma kehitettiin .NET Frameworkilla, koska siitä oli aikaisempaa positiivista kokemusta opinnoissa. Tietopohjana oli .NET-kurssi, jonka aikana toteutettiin ASP.NET-verkkosovellus C#-kielellä.

.NET Framework on Microsoftin kehitysalusta, jonka avulla voidaan rakentaa sovelluksia Windows-ympäristöön. (Microsoft Developer Network c.) .NETin tärkeimpiin ominaisuuksiin kuuluu kielten yhteen toimivuus (engl. language interoperability). Ohjelmisto voidaan siis luoda millä tahansa .NET-ohjelmointikielellä, joita ovat esimerkiksi C#, Visual Basic ja Visual C++. Kaikki .NET-kielet ovat oliosuuntautuneita ja kaikilla .NET-kielillä on yhteinen ajoympäristömoottori (engl. runtime engine). (Connolly 2007, 9.) .NET-kielillä on kattava perusluokkakirjasto (engl. base class library), joka tarjoaa yhtenäisen objektimallin kaikille .NET-kielille. (Troelsen 2008, 6.)

.NET-alusta rakentuu kolmen kokonaisuuden varaan: CLR, CTS ja CLS (Kuvio 1). Ohjelmoijan kannalta .NET voidaan nähdä ajoympäristönä (engl. runtime environment) ja kattavana perusluokkakirjastona. Ajoympäristökerroksesta puhutaan myös nimellä Common Language Runtime eli CLR. Sen päätarkoitus on paikantaa, ladata ja hoitaa .NETin tyyppejä ohjelmoijan puolesta. Common Type System eli CTS-määritelmä kuvaa täydellisesti kaikki mahdolliset tyypit, joita ajoympäristö tukee, ja määrittelee, miten nämä kokonaisuudet ovat keskenään vuorovaikutuksessa. CTS myös vastaa siitä, miten niiden yksityiskohdat esitetään .NETin metadataformaatissa. CLS eli Common Language Specification määrittelee yhteiset tyypit ja ohjelmointirakenteet (engl. programming construct), joita kaikki .NET-ohjelmointikielet käyttävät. Jos siis ohjelmoija käyttää jotakin tyyppiä, minkä CLS ymmärtää, kaikki muutkin .NET-kielet ymmärtävät kyseisen tyypin. (Troelsen 2008, 6-7.)



Kuvio 1. CLR:n, CTS:n, CLS:n ja perusluokkakirjastojen suhteet. (Troelsen 2008, 7.)

2.3.1 Windows Forms

Windows Forms on .NET Frameworkin sisältämä ohjelmointirajapinta, jolla voidaan toteuttaa graafisia käyttöliittymiä Windowsin työpöytäsovelluksiin, ja sitä käytettiin tämän sovelluksen toteuttamisessa. .NET Framework julkaistiin vuonna 2001, ja siitä lähtien luokkakirjastot ovat sisältäneet Windows Forms APIn. (Troelsen 2008, 955.) Windows Forms API koostuu sadoista tyypeistä (luokista, rajapinnoista, rakenteista, numeroiduista tyypeistä (engl. enums) ja delegaateista, jotka on järjestetty lukuisiin System.Windows.Forms.dll:n nimiavaruuksiin. Tärkein näistä nimiavaruuksista on System.Windows.Forms. Sen tyypit voidaan jaotella neljään laajaan kategoriaan: ydininfrastruktuuriin, kontrolleihin, komponentteihin ja yleisiin dialogi-ikkunoihin. Ydininfrastruktuurin tyypit edustavat Windows Forms -ohjelman ydinoperaatioita kuten Form tai Application. Kontrolleita käytetään käyttöliittymien rikastamiseen, ja ne kaikki periytyvät Control-perusluokasta. Kontrolleita ovat esimerkiksi DataGridView, MenuStrip ja Button. Komponentit, kuten esimerkiksi Tooltip ja ErrorProvider, tarjoavat visuaalisia ominaisuuksia Windows Forms -ohjelmaan. Yleiset dialogi-ikkunat tarjoavat lukuisia eri dialogi-ikkunoita yleisimmille operaatioille kuten OpenFileDialog. (Troelsen 2008, 955-956.)

Varteenotettava vaihtoehto Windows Forms API:lle oli WPF (Windows Presentation Foundation). Windows Formsiin päädyttiin, koska siitä oli hieman aikaisempaa

kokemusta. Toinen syy Windows Formsin valitsemiseen on siihen löytyvät laajat, Microsoftin tarjoamat tuki- ja oppimisresurssit. WPF olisi ollut täysin uusi teknologia, jonka opetteluun olisi luultavasti mennyt paljon aikaa.

2.3.2 C#-ohjelmointikieli

C# on Microsoftin kehittämä ohjelmointikieli .NET Frameworkille (ECMA International 2006, 21.). Sovellus kehitettiin C#-kielellä, koska ohjelmointikielenä C# oli tuttu. Kielenä C# on moderni, olio-orientoitunut ja tyyppiturvallinen (engl. type-safe). C#:ssa yhdistyy nopean kehityksen mallin (RAD) kielten korkea tuottavuus ja C++:n teho. C#-kielisen ohjelman lähdekoodi säilötään tyypillisesti .cs-tiedostoihin. (ECMA International 2006, 37.)

2.4 Microsoft Visual Studio Express

Visual Studio on Microsoftin kehittämä IDE, joka sisältää koodieditorin, sisäänrakennetun debuggerin sekä kone- että lähdetason debuggaamiseen, graafisen suunnittelunäkymän käyttöliittymien rakentamiseen ja muita tärkeitä työkaluja. Sovellus kehitettiin Visual Studiolla, koska se oli ennestään tuttu ja Microsoft tarjoaa Visual Studio –kehitysympäristölle erittäin hyvän tuen esimerkiksi Microsoft Developer Networkin kautta.

Sovelluksen tuottamiseen käytettiin Visual Studion 2012 & 2013 Express for Desktop -versioita. Ohjelman for Desktop -versio on tarkoitettu erityisesti Windows-työpöytäsovellusten luomiseen. Visual Studion Express-versiossa ei ole yhtä paljon ominaisuuksia kuin maksullisissa Visual Studion versioissa, mutta opinnäytetyöprojektiin Express-versio osoittautui riittäväksi.

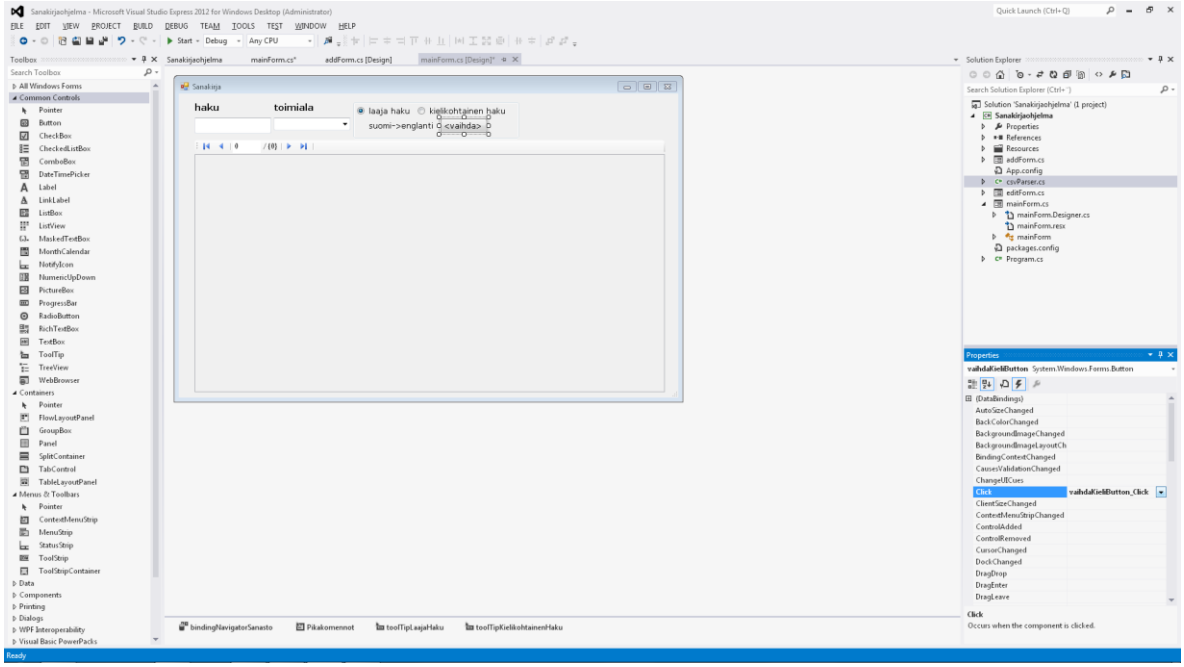
2.4.1 Visual Studion käyttöliittymä

Visual Studion käyttöliittymä rakentuu monille ikkunoille, joita voi siirrellä ja muokata melko vapaasti. Oletuksena iso valkoinen pääikkuna, jossa koodin editointi tapahtuu, on keskellä ruutua. Pääikkunan oikealla puolella ruudun yläkulmassa on Solution

Explorer-ikkuna, josta näkee projektin tiedostorakenteen. Solution Explorerilla voi myös avata tiedostoja.

Solution Explorerin alapuolella, ruudun vasemmassa alakulmassa on Properties-ikkuna, jonka avulla pystyy tarkastelemaan yksittäisten kontrollien asetuksia, arvoja ja toiminnallisuuksia. Properties-ikkunaa käytetään suunnittelunäkymän tukena. Suunnittelunäkymän kontrollien ominaisuuksia ja toimintoja voi asettaa ja muokata Properties-ikkunan Events- ja Properties-välilehdillä. (Microsoft Developer Network d.)

Pääikkunan vasemmalla puolella on Toolbox, jota käytetään suunnittelunäkymässä. Toolbox sisältää lukuisia eri kontrolleja, esimerkiksi painikkeita, valikoita ja tekstikenttiä, joita voidaan tuoda suunnittelunäkymään raahaamalla. Suunnittelunäkymällä toteutetaan sovelluksen käyttöliittymä. (Microsoft Developer Network e.)



Kuva 2. Visual Studio 2013 Express -käyttöliittymässä on auki mainForm.cs:n suunnittelunäkymä.

2.4.2 IntelliSense ja XML-dokumentaatio

IntelliSense on Visual Studion ominaisuus, joka tarjoaa käyttäjälle loogisia koodielementtejä pudotusvalikkomuodossa. Ominaisuuden tarkoitus on auttaa sovelluksen kehittäjää antamalla tälle mahdollisuus tuottaa automatisoitua koodia Visual Studion koodieditorissa. Käyttäjä säästää ominaisuuden avulla koodin kirjoittamiseen kuluvaa aikaa ja välttyy myös kirjoitusvirheiltä. (Microsoft Developer Network f.)

XML (Extensible Markup Language) on joukko tekstiformaattien suunnittelusääntöjä, joiden avulla voidaan jäsentää tietoa. XML:n avulla tietokoneiden on helppo tuottaa ja lukea tietoa sekä varmistaa, että tietorakenteet ovat yksiselitteisiä. (W3C 2001.)

Visual Studio ja IntelliSense tukevat XML-koodidokumentaatiota kaikille .NET-kielille, mukaan lukien C#. (Elliott 2010.) Koodi voidaan dokumentoida sisällyttämällä XML-elementit niille tarkoitettuihin kommenttikenttiin (Esimerkki 1), jotka luodaan automaattisesti käyttäjän kirjoittaessa kolme kauttaviivaa ennen koodilohkoa, joihin kommentit viittaavat. (Microsoft Developer Network h.)

```
/// <summary>
///  Tämä luokkaa suorittaa tärkeän funktion.
/// </summary>
public class MinunLuokka { }
```

Esimerkki 1. automaattisesti luotu XML-elementti <summary> kuvaa MinunLuokka-luokan toimintaa.

Visual Studio jäsentää projektissa käytetyt .cs-tiedostot, jotta se voi tuottaa XML-dokumentaatiotiedoston. Automaattisesti XML-dokumentaatiotiedostoa ei tuoteta. Käyttäjän täytyy laittaa kyseinen ominaisuus päälle, antaa tuotettavalle tiedostolle nimi ja määritellä tulevan tiedoston sijainti. Tämän pystyy tekemään Solution Explorerin Project-välilehdellä. Varsinainen XML-dokumentaatiotiedosto tuotetaan vasta solutionin luomisen (engl. build) yhteydessä. (Elliott 2010.)

2.5 Tietokannan valinta

Koska sovelluksella piti pystyä lisäämään ja muokkaamaan sanoja ja haun piti olla tehokas, oli selvää, että tarvittiin tietokantapohjainen ratkaisu. Tietokannan valitsemiseen vaikutti muutama tärkeä tekijä. Toimeksiantajan kannalta oli ensinnäkin toivottavaa, että tietokannasta ei koituisi taloudellisia kuluja. Koska tietokanta ja sovellus oli suunniteltu vain yhdelle käyttäjälle, tulimme siihen tulokseen, ettei erillinen palvelin tietokantaa varteen ole järkevä ratkaisu. Palvelinta täytyisi pitää yllä ja se veisi sekä rahaa että aikaa, koska käyttäjällä ei itse ole valmiuksia tietokannan ylläpitämiseen ja hallinnointiin.

Tärkeää oli myös mahdollisen asennusprosessin helppous. Oli suotavaa, ettei käyttäjän tarvitsisi erikseen asentaa tietokantaa koneelleen. Nämä vaatimukset siis pudottivat pois kaikki maksulliset ja erillisen palvelimen vaativat tietokannat. Tässä vaiheessa vaihtoehtoja oli muutama. Esimerkiksi Microsoft SQL Serverin Compact Edition ja Oracle Database Express Edition täyttivät vaatimukset kohtalaisesti, mutta ne vaativat erillisen asennuksen. Lisäksi ne sisältävät palvelimen, jonka täytyisi vähintäänkin olla tietokoneen taustalla käynnissä prosessina, aina kun sovellusta tultaisiin käyttämään.

SQLite-tietokanta sopi sovelluksen tarkoituksiin loistavasti, koska se ei vaadi erillistä asennusta käyttäjän koneelle. Tämä tarkoitti sitä, että tietokanta voitiin tehdä ohjelmallisesti, jolloin sovellus on hyvin helppo siirtää tietokoneelta toiselle. Jos tietokonetta jouduttaisiin jostain syystä vaihtamaan esimerkiksi pöytäkoneesta kannettavaan tietokoneeseen, riittää, että ohjelman asennuskansio kopioidaan koneesta toiseen.

Työssä tietokannaksi valittiin SQLite, joka on täysin ilmainen avoimen lähdekoodin relaatiotietokantaratkaisu. SQLite täyttää tietokantavaatimukset täydellisesti, koska se on ilmainen, ei vaadi erillistä asennusta eikä palvelinta. SQLite eroaa useimmista muista SQL-tietokannoista siinä, ettei se käytä erillisiä serveriprosesseja. SQLiten tietokanta on yksittäinen, tietokoneen kovalevyllä sijaitseva tiedosto, joka sisältää kaikki tarvittavat tietokannan tiedot: taulujen rakenteet, niitä koskevat ohjeet ja asetukset sekä taulujen sisällöt. (SQLite a.) SQLite-kirjasto kaikkine ominaisuuksineen vie kovalevyltä tilaa vain noin 400KB ja kun tarpeettomat ominaisuudet poistetaan käytöstä, kirjaston koko on noin 200KB. (SQLite d.) SQLite on kirjoitettu ANSI-C:llä, ja se käyttää ainoastaan seitsemää C-standardikirjaston funktiota. (SQLite f.)

Jokaisella SQLite-tietokannan sarakkeella on oma tyyppiaffiniteettinsa. Jokaiseen sarakkeeseen voidaan säilöä minkä tyyppistä tietoa tahansa, mutta jos mahdollista, sarakkeet käyttävät tiettyä varastointiluokkaa. Tietyn sarakkeen mieluisin varastointiluokka on sen affiniteetti. Se voi olla joko TEXT, NUMERIC, INTEGER, REAL tai NONE. Esimerkiksi TEXT-affiniteettinen sarake varastoi kaiken tiedon käyttämällä hyväkseen varastointiluokkia NULL, TEXT tai BLOB. Jos tähän sarakkeeseen lisätään numeerista tietoa, muutetaan tieto tekstityyppiseksi ennen sen varastointia. SQLite ei aseta pituusrajoituksia merkkijonojen, binääritiedon (BLOB) tai numeroiden arvoille. Esimerkiksi CREATE TABLE -lauseen yhteydessä käytetty tyyppi VARCHAR(255) muuttuu yksinkertaisesti TEXT-affiniteetiksi, joka varaa tilaa muistista ainoastaan sen verran kuin on tarpeen. VARCHAR(255) ei siis varaa tilaa 255 merkille, vaan juuri sen verran kuin merkkejä sijoitetaan. 255 ei myöskään ole merkkijonon maksimipituus, koska SQLite käyttää pituusrajoituksena ainoastaan globaalia SQLITE_MAX_LENGTH -rajoitusta, joka on oletusarvoisesti miljardi merkkiä. (SQLite c; SQLite e.)

SQLite ei tue tallennettujen proseduurien käyttöä. (SQLite b.) Tallennettujen proseduurien käyttö ei sovelluksen kannalta ollut kuitenkaan tarpeellista, koska operaatiot, joita tallennettujen proseduurien avulla oltaisiin voitu sovelluksessa tehdä, esimerkiksi monen tietueen samanaikainen päivittäminen viite-eheyden ylläpitämiseksi, onnistuu myös liipaisimien (engl. trigger) avulla. Jos käyttäjä esimerkiksi poistaa toimialan, olisi tallennettu proseduuri voinut pitää tietokannan viite-eheyden kunnossa poistamalla toimialan niistä sanapareista, joissa toimialaa oli käytetty. Liipaisimen avulla voidaan kuitenkin tässä tapauksessa saavuttaa sama lopputulos. (Esimerkki 2)

Liipaisimet ovat tietokantaoperaatioita, jotka toteutetaan automaattisesti tietyn tietokantatapahtuman sattuessa. Liipaisin voi esimerkiksi laueta, kun DELETE-, INSERT- tai UPDATE-komento tapahtuu tietyn tietokantataulun yhteydessä. (SQLite g.) Ero tallennettujen proseduurien ja liipaisimien välillä on se, että liipaisimet laukeavat tietyn tapahtuman yhteydessä, kun taas tallennettuja proseduureja täytyy kutsua sovelluksesta. Liipaisimien hyvä puoli SQLite-tietokannassa oli, että ne voitiin luoda ohjelmakoodissa niin kuin tietokannan taulutkin.

```
"CREATE TRIGGER IF NOT EXISTS \"trgDeleteToimiala\" AFTER DELETE ON \"Toimialat\"
BEGIN UPDATE \"Sanasto_fi_en\" SET \"toimiala\" = null WHERE \"toimiala\" =
old.toimiala; END";
```

Esimerkki 2. Esimerkissä liipaisin, joka toimialan poiston jälkeen päivittää sanasto_fi_en-taulussa kyseisen toimialan sisältäneiden tietueiden toimialakenttien arvot tyhjiksi.

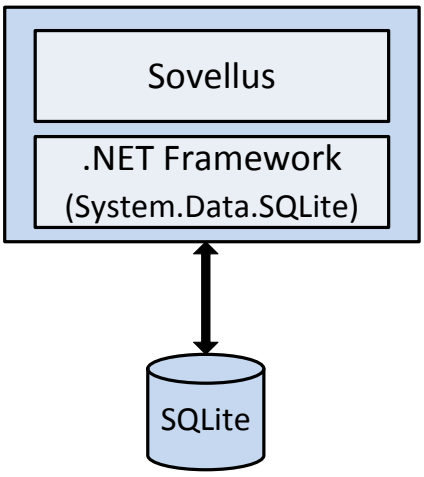
3 Projektin suunnittelu, toteutus ja kehitys

Projekti jakautui kolmeen vaiheeseen, joissa jokaisessa oli tarkoitus saada tiettyjä ohjelman toimintoja valmiiksi priorisoinnin perusteella. Projektin kehitysmalli perustui inkrementaaliseen kehitysmalliin, jonka perusidea on se, että ohjelman ensimmäinen versio on yksinkertainen ja pitää sisällään vain kriittisimmät ja korkeapriorisoidut toiminnot. Ominaisuuksia lisätään sovelluksen tulevissa versioissa projektin edetessä. (Janssen) Projekti piti sisällään seurantakokouksia ja päättökokouksen, joiden ajankohdat määrittelivät projektin vaiheiden käynnistymisen ja lopetuksen. Tarkemmat tiedot siitä, mitä toteutettiin milloin, löytyy liitteestä 1.

Sovelluksessa käytetyt tekniikat ja suunnitteluratkaisut olivat ainoastaan työvälineitä hyvän käytettävyyden saavuttamiseksi, mikä oli sovelluksen käytön kannalta päätarkoitus.

3.1 Sovellusarkkitehtuuri

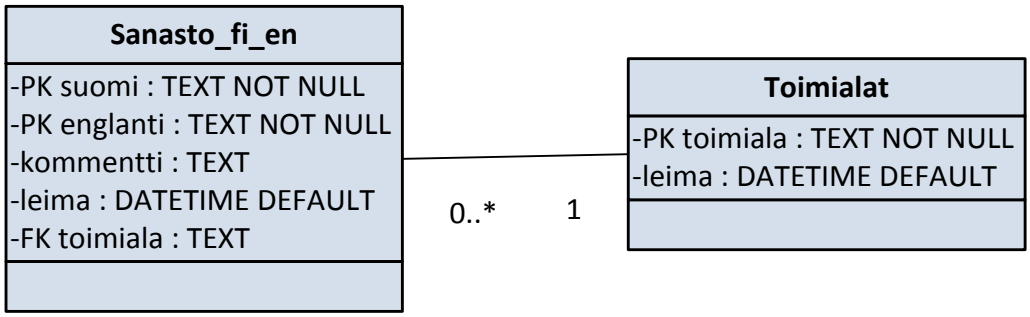
Sovelluksen arkkitehtuuri vastaa melko tarkasti kaksitasoarkkitehtuuria. (Kuvio 2) Kaksitasoarkkitehtuuri on hyvä ratkaisu pieniin sovelluksiin, joissa ei ole paljon lomakkeita. (Sheriff 2002.) Sovellus kommunikoi SQLite-tietokannan kanssa System.Data.SQLite .NET -ajurin avulla.



Kuvio 2. Sovellus kommunikoi tietokannan kanssa .NET Frameworkin välityksellä.

3.2 Tietokannan suunnittelu ja toteutus

Tietokannan suunnittelua ja toteutusta ohjasi käytettävyys. Tärkeintä oli, että tietokantaratkaisu mahdollistaisi sovelluksen nopean käytön ja pystyisi tarjoamaan toimeksiantajalle hänen haluamansa ominaisuudet. Tämän takia relaatiotietokantaa ei ole normalisoitu, koska se voisi hidastaa sovelluksen toimintanopeutta (Chapple) Tietokantaratkaisu on hyvin yksinkertainen. Tauluja on kaksi: Sanasto_fi_en ja Toimialat. Sanasto_fi_en-taulussa on tallessa sanaparit ja niihin liittyvät kommentit. Toimialat-taulu sisältää käyttäjän lisäämät toimialat. Leima-kenttä on automaattisesti luotu aikaleima.



Kuvio 3. Kuvio havainnollistaa tietokannan rakennetta. Tietokanta koostuu kahdesta taulusta: Sanasto_fi_en ja Toimialat.

Tietokanta luodaan ohjelmallisesti sovelluksen ajon yhteydessä metodien (esimerkki 3) createTables() ja ExecuteQuery() avulla. Tietokannan ohjelmallinen luonti mahdollistaa sovelluksen siirrettävyyden tietokoneesta toiseen. Tällöin tietokantaa ei tarvitse erikseen luoda, vaan sovellus luo tietokannan automaattisesti sovelluksen käynnistyksen yhteydessä.

```
/// <summary>
/// Luodaan taulut, indeksit ja triggerit, mikäli ei jo olla luotu.
/// </summary>
public void createTables()
{
    String createSQL = "CREATE TABLE IF NOT EXISTS Toimialat (toimiala TEXT
    NOT NULL PRIMARY KEY, leima DATETIME DEFAULT current_timestamp)";
    ExecuteQuery(createSQL);
    createSQL = "CREATE TABLE IF NOT EXISTS Sanasto_fi_en (suomi TEXT NOT
    NULL , englanti TEXT NOT NULL, toimiala TEXT, kommentti TEXT, leima
    DATETIME DEFAULT current_timestamp, FOREIGN KEY (toimiala) REFERENCES
    Toimialat(toimiala), PRIMARY KEY (suomi, englanti))";
    ExecuteQuery(createSQL);
}
```

```
/// <summary>
/// Suorittaa parametrissa txtQuery annetun SQL-komennon.
/// </summary>
/// <param name="txtQuery">Suoritettava SQL-komento</param>
public void ExecuteQuery(string txtQuery)
{
    SetConnection();
    sql_con.Open();

    sql_cmd = sql_con.CreateCommand();
    sql_cmd.CommandText = txtQuery;
    try
    {
        sql_cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        if (ex.Message.Contains("not unique") == false)
        {
            MessageBox.Show(ex.Message);
        }
    }

    sql_con.Close();
}
```

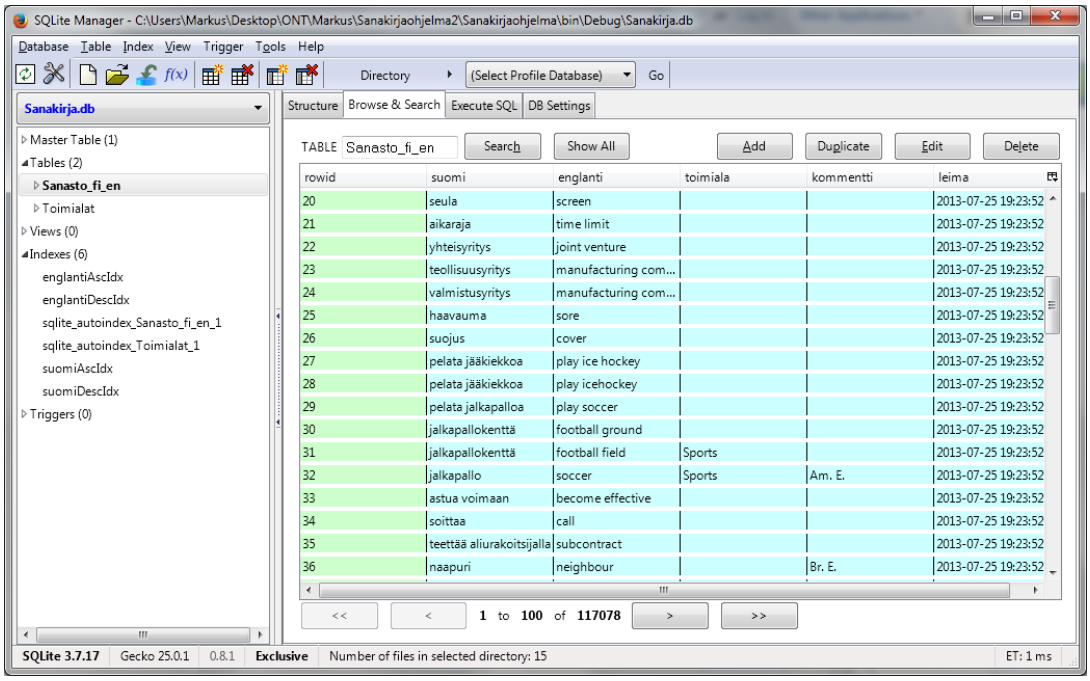
Esimerkki 3. ExecuteQuery()-metodi suorittaa parametrissa txtQuery annetun SQL-komennon, ja createTables()-metodi luo tietokantaan taulut ExecuteQueryn avulla, mikäli niitä ei olla luotu.

Kuten aikaisemmin on mainittu, SQLite-tietokanta sijaitsee kiintolevyllä, eikä tietokanta ole internetiin missään yhteydessä. Tietokantaan liittyvät tietoturvakysymykset ovat siis lähinnä triviaaleja. Käyttöjärjestelmä on salasanan takana, eli ainakin periaatteessa vain ohjelman oikealla käyttäjällä on pääsy sovellukseen. Kuitenkin esimerkiksi sovelluksen käytön yhteydessä avautuvat SQL-injektointimahdollisuudet on valmiiksi poistettu parametrisoiduilla SQL-kyselyillä, jos sovellus haluttaisiin esimerkiksi myöhemmin siirtää web-pohjaiseksi. (Microsoft Technet) (Esimerkki 4)

```
"INSERT INTO Sanasto_fi_en (suomi,englanti,toimiala,kommentti) VALUES
 (@suomi, @englanti, @toimiala, @kommentti)";
mainForm.sql_cmd.Parameters.AddWithValue("@suomi", mainForm.sana.suomi);
 mainForm.sql_cmd.Parameters.AddWithValue("@englanti",
 mainForm.sana.englanti);
mainForm.sql_cmd.Parameters.AddWithValue("@kommentti",
 mainForm.sana.kommentti);
mainForm.sql_cmd.Parameters.AddWithValue("@toimiala",
 mainForm.sana.toimiala);
```

Esimerkki 4. Sanasto_fi_en-taulun SQL INSERT -lause on parametrisoitu, jotta SQL-injektointi ei onnistuisi.

Projektissa tietokannan testaukseen käytettiin SQLite Manageria. SQLite Manager on lisäosa Mozilla Firefox -selaimeen. SQLite Managerilla voidaan selailla, luoda ja poistaa tauluja, näkymiä, indeksejä ja triggereitä. Olemassa oleviin tauluihin voidaan esimerkiksi lisätä sarakkeita tai rivejä, ja niitä voidaan myös poistaa. SQLite Manageriin voidaan kirjoittaa SQL:ää ja tarkastella kyselyiden tuloksia. SQLite Manageriin pääsee käsiksi Mozilla Firefox -selaimen Web Developer -valikon kautta. (SQLite Manager)



Kuva 3. SQLite Managerissa tarkastellaan taulun Sanasto_fi_en sisältöä.

3.3 Käyttöliittymän suunnittelu ja toteutus

Käyttöliittymän suunnittelu ja toteutus tapahtuivat jokseenkin samanaikaisesti ominaisuuksien toteuttamisen yhteydessä. Vaatimukset keskittyivät lähinnä toiminnallisuuksiin, ei niinkään itse käyttöliittymään. Yksi vaatimus esimerkiksi oli, että sanastoon täytyy pystyä lisäämään sanoja helposti ja nopeasti.

Sovelluksen käyttöliittymää suunniteltaessa olikin tärkeää tietää, miten käyttäjä on tottunut työskentelemään. Uuden käyttöliittymän toteutuksessa pyrittiin vertaamaan uutta käyttöliittymää nykyisiin käytössä oleviin ratkaisuihin ja tunnistamaan käyttöliittymien hyvät ja huonot puolet. Suoraa vertailukohtaa ei ole, mutta vertailussa keskityttiin ohjelmiin, joita sovelluksen tuleva käyttäjä hyödyntää nyt työssään. Tärkeimmät näistä olivat elektroniset MOT-, Webster-, SOED,- Collins- ja WSOY-sanakirjat. Näiden lisäksi käyttäjä hyödyntää internetin eri palveluita, esimerkiksi yliopistojen tarjoamia erikoissanastoja ja sanakirjoja.

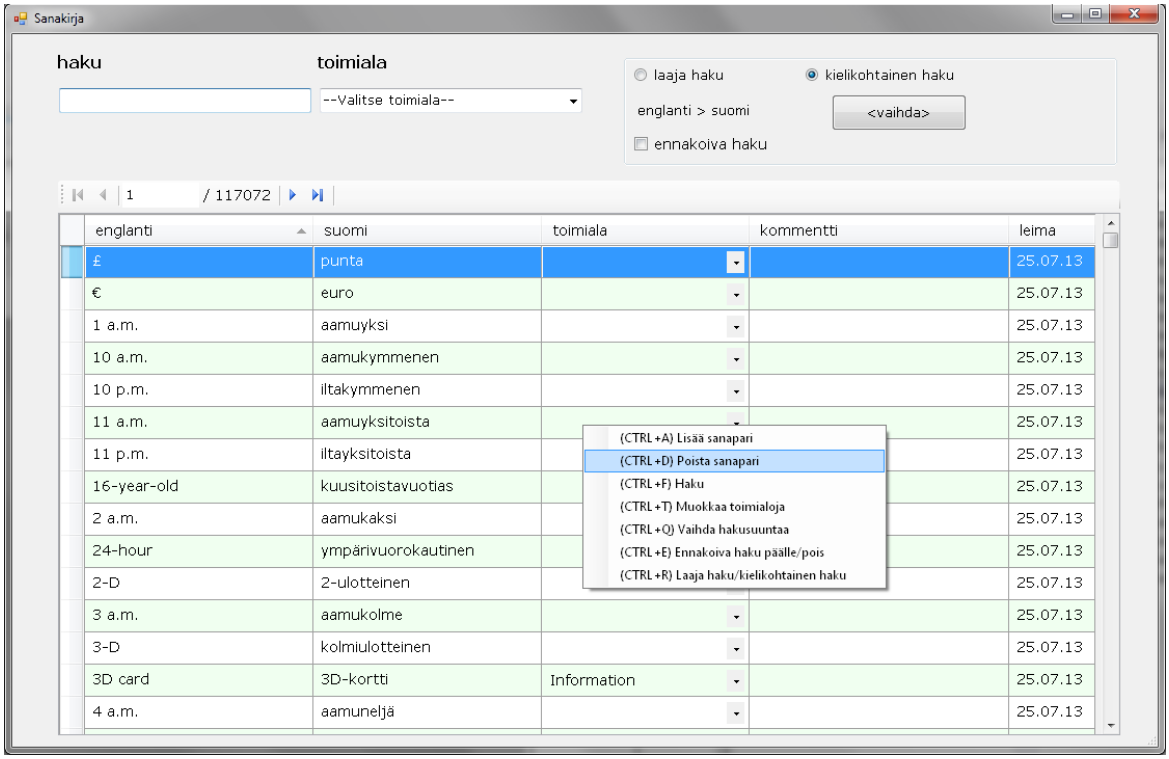
3.3.1 Pikakomentojen käyttö ja sovelluksen toiminta

Suurin käyttöliittymällinen toive (varsinaista vaatimusta tästä ei kirjattu) oli hiiren käytön minimointi. Sovelluksessa käytetään pikanäppäimiä, jotka toimivat Control-näppäinyhdistelmillä. Esimerkiksi CTRL+F-pikakomento saa hakukentän aktiiviseksi ja tyhjentää sen, jotta käyttäjä voi heti alkaa kirjoittaa uutta hakua ilman hiiren siirtelyä tai klikkailua. Pikanäppäinten valinnassa otettiin huomioon toimeksiantajan toiveet ja yleiset Windows-ympäristön pikanäppäinstandardit. Vaikka toimeksiantajalle pikanäppäinten loogisuus ei ollut kynnyskysymys, päästiin loogisuuden kannalta useimmiten hyvään lopputulokseen.

Toimeksiantaja mietti, mitkä pikanäppäimet olisivat helposti muistettavissa ja sormien ulottuvilla. Päädyttiin Control-näppäinyhdistelmiin. CTRL+A on pikanäppäin uuden sanaparin lisäykselle (Add), CTRL+D:llä poistetaan sanapari (Delete) ja CTRL+F:llä päästään suoraan tekemään uusi haku (Find, universaali haun tai etsimisen pikakomento). CTRL+T:llä päästään toimialojen hallintaan. Edellä mainittujen tärkeimpien ohjelman toimintojen lisäksi pikanäppäimien avulla voidaan vaihtaa hakutyyppejä ja hakusuuntaa kielestä toiseen. Esimerkiksi CTRL+Q vaihtaa

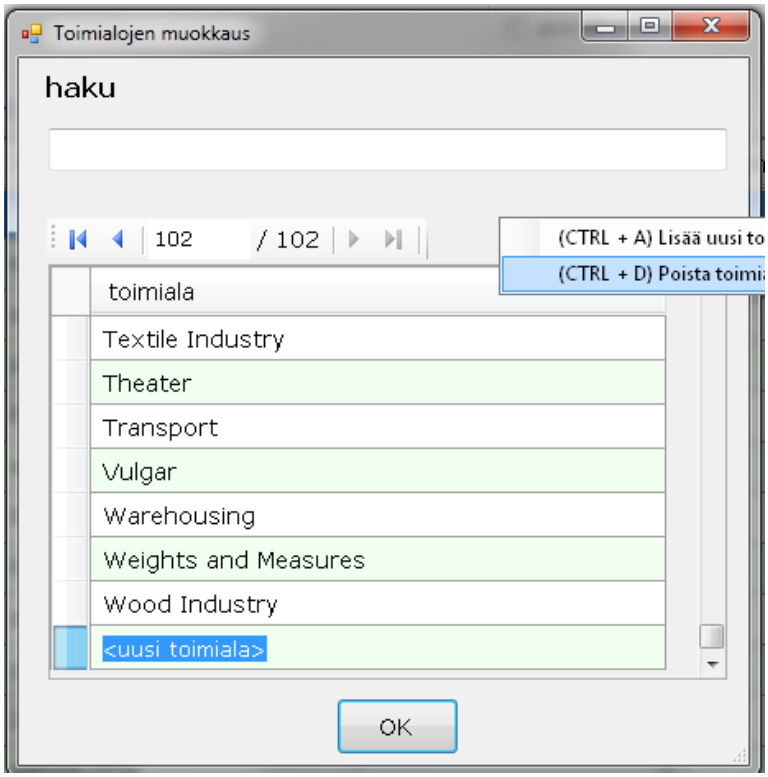
hakutyyppiä laajan ja kielikohtaisen haun välillä ja CTRL+R vaihtaa hakusuuntaa. Näiden kahden pikanäppäimen muistettavuus lienee kehno, koska näppäinyhdistelmä ei ole looginen. Pikanäppäimien valinnassa pyrittiin välttämään yhteentörmäyksiä yleisessä käytössä olevien pikanäppäinyhdistelmien kanssa. Esimerkiksi kopiointi (CTRL+C), liimaa (CTRL+V) ja kumoa (CTRL+Z) toimivat sovelluksessa normaaliin tapaan.

Käyttöliittymässä hyödynnetään Windows Formsin DataGridView-kontrollia, jonka avulla käyttäjälle näytetään hakutuloksia. Sen kautta myös muokataan, poistetaan ja lajitellaan olemassa olevaa tietoa. DataGridView-kontrollin soluissa tapahtuvat muutokset päivittyvät tietokantaan aina, kun käyttäjä lähtee pois aktiivisesta solusta, ja vain, jos solussa on tapahtunut muutoksia. Käytännössä muutos siis tapahtuu joko enteriä painamalla, jolloin solun fokus automaattisesti vaihtuu alapuolella olevaan soluun, tai aktivoimalla hiirellä toinen solu. DataGridView-kontrollissa voidaan navigoida nuolinäppäinten ja tabulaattorin avulla. Sanojen lisäys ja toimialojen hallinta tapahtuu omissa lomakkeissaan. Tietyn sanaparin toimialaa voi kyllä muuttaa DataGridView-kontrollin kautta. Kun käyttäjä klikkaa sovellusta hiiren oikealla painikkeella, aukeaa pieni ponnahdusvalikko (Kuva 4), josta hän voi suorittaa haluamansa komennot joko klikkaamalla tai pikanäppäimen avulla. Ponnahdusvalikko toimii myös muistutuksena käyttäjälle pikakomentojen näppäinyhdistelmistä. Toinen vaihtoehto olisi ollut tehdä erillinen valikko vaikkapa perinteisesti sovelluksen vasempaan ylälaitaan.

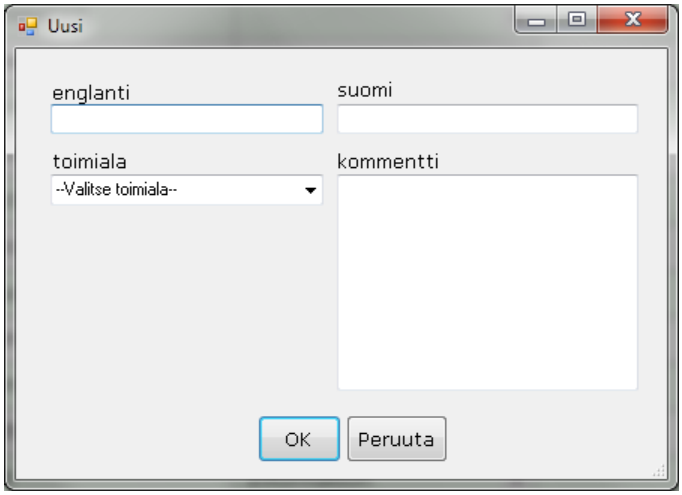


Kuva 4. Sanastotyökalun ponnahdusvalikko toimii käyttäjälle muistutuksena pikanäppäinyhdistelmistä. Menusta voidaan myös hiirellä valita haluttu toiminto.

Toimialahallinnan ikkunassa toimivat samat näppäinyhdistelmät CTRL+A ja CTRL+D. Toimialahallinta-ikkunassa voidaan poistaa, lisätä ja muokata toimialoja (Kuva 5). Sanapareja voidaan lisätä omassa ikkunassaan. Sanaparille voidaan määrittää toimiala olemassa olevasta pudotusvalikosta, jos halutaan. Uusi toimiala on myös mahdollista lisätä sanaparin lisäämisen yhteydessä kirjoittamalla toimialakenttään uuden toimialan nimen. Samassa yhteydessä käyttäjällä on mahdollisuus lisätä sanaparia vastaava kommentti, jossa voi olla esimerkkilause ja asiakkaan tai työn nimi (Kuva 6).



Kuva 5. Toimialahallintaikkunassa voidaan poistaa, muokata ja lisätä toimialoja.



Kuva 6. Uuden sanaparin lisäämisen yhteydessä voidaan sille määrittää toimiala ja lisätä kommentti.

3.3.2 Hakutyypit

Sovelluksessa on käytössä kaksi erilaista hakua ja yksi valinnainen hakutyyppi, jota voidaan käyttää kummankin haun yhteydessä. Käyttäjä voi hakea laajalla tai kielikohtaisella haulla. Laaja haku hakee suomi-, englanti- ja kommenttikentän perusteella. Kielikohtainen haku hakee ainoastaan toisen kielen perusteella. Hakusuunta

näkyy omana tekstikenttänään, ja hakusuuntaa voidaan vaihtaa klikkaamalla vaihda-nappia tai pikanäppäinyhdistelmä CTRL+Q:n avulla. Haku tapahtuu aktivoimalla hakukenttä joko sitä klikkaamalla tai käyttämällä CTRL+F-pikanäppäinyhdistelmää. Käyttäjä kirjoittaa haluamansa sanan, ja voi halutessaan rajoittaa haun koskemaan tiettyä toimialaa valitsemalla toimialan pudotusvalikosta. Haku suoritetaan enteriä painamalla. Tämän jälkeen hakutulokset päivittyvät DataGridView-kontrolliin. Ennakoiva haku oli yksi sovelluksen vaatimuksista ja se toteutettiin valinnaiseksi hakutyypiksi. Ennakoiva haku päivittää DataGridView-kontrollia aina, kun hakukentän sisältö muuttuu. Ennakoivaa hakua voidaan käyttää sekä laajan haun että kielikohtaisen haun yhteydessä. Kun käyttäjä siirtää kursorin laajan tai kielikohtaisen haun päälle, muistutetaan häntä kyseisen haun toiminnasta tooltipin avulla.

Lisäksi käyttäjä voi suorittaa päivämäärähaun kirjoittamalla päivämäärän hakukenttään, jolloin haetaan suoraan leima-kentästä, mikä on käytettävyyden kannalta hyvä ratkaisu. Käyttäjä ei joudu silloin erikseen määrittelemään päivämäärähakua. (Esimerkki 5 luvussa 4.1)

3.4 Ylläpidettävyys

Projektissa hyödynnettiin Visual Studion tarjoamaa mahdollisuutta tuottaa XML-dokumentaatio sovelluksesta. Koodi kommentoitiin Visual Studion sisäänrakennetulla XML-kommentointiominaisuudella. Näin sovelluksesta saatiin XML-tiedosto, joka kuvasi ohjelmakoodin toiminnan.

Dokumentaation tarkoituksena on auttaa sovelluksen mahdollista tulevaa kehittäjää ymmärtämään kirjoitettu koodi, jolloin sovelluksesta tulee ylläpidettävämpi.

4 Lopputulokset ja johtopäätökset

4.1 Ongelmat ja niiden ratkaiseminen

Databexit Oy omistaa n. 117 000:n sanaparin sanaston, jota oli osittain kategorisoitu toimialojen mukaan ja kommentoitu. Kyseisen sanaston sisältö tuotiin ohjelman tietokantaan ohjelmallisesti. Sanasto toimi loistavasti testidatana ohjelman ominaisuuksien testaamisessa, ja antoi myös arvokasta tietoa ohjelman käytettävyydestä. Tietokanta, jossa on 117 000 sanaparia, on melko suuri. Suurella tietokannalla tehdyt kokeilut auttoivat ymmärtämään ohjelmaan tehtyjen muutosten merkitystä.

Jotkut ohjelmakoodiin tehdyt muutokset näkyivät välittömästi sanahauissa joko hitautena tai nopeutena. Yksi isoimmista haasteista ohjelman nopeuden kannalta oli ennakoiva haku. Kun hakukenttään kirjoittaa, päivittyy ohjelman DataGridView-kontrolli jokaisen kentässä tapahtuvan muutoksen jälkeen. Aina kun DataGridView-kontrollia päivitetään hakuehtojen perusteella, ilmenee viive haetun datan suuruudesta riippuen. Viive tuntuu nimenomaan hakurivillä, kun merkkejä lisätään tai poistetaan, koska jokaisen muutoksen kohdalla suoritetaan haku. Esimerkiksi kun laajassa haussa kirjoitetaan hakuriville e-kirjain, haetaan kaikki rivit, joissa jokin kenttä sisältää e-kirjaimen. Mikäli kielikohtainen haku on käytössä, haetaan valitun kielisuunnan mukaisesti ainoastaan kenttiä, jotka alkavat e-kirjaimella. Tällöin haku on nopeampi.

Haut tapahtuvat doFind()-metodilla (esimerkki 5), joka haarautuu vielä "alimetodeihin". Metodin perusteella käyttäjälle näytetään hakukriteereiden mukainen tieto. Jos käyttäjä esimerkiksi hakee kielikohtaisella haulla suunnasta suomi->englanti, käytetään doFindSuomi()-metodia.

```
public void doFind()
        {
            if (okToFind)
            {
                okToFind = false;
                IFormatProvider culture =
                         new System.Globalization.CultureInfo("fi-FI", true);
                isDate = true;
                try
                {
                    toFindDate = DateTime.Parse(textBoxFind.Text, culture,
                         System.Globalization.DateTimeStyles.AssumeLocal);
                }
                catch
                {
                    isDate = false; ;
                }
                loading = false;

                if (isDate)
                {
                    doFindDate();
                }
                else
                {
                    if (laajaHakuRadioButton.Checked)
                        doFindAll();
                    else
                        if (suomiEnglanti == true)
                            doFindSuomi();
                        else
                            doFindEnglanti();
                }
                asetaGrid();
                vaihdaSarakkeet();
            }
        }
```

Esimerkki 5. doFind()-metodi haarautuu ”alimetodeihin”.

Aluksi haut veivät melko kauan, koska jokaisen haun yhteydessä koko DataGridView-kontrolli päivitettiin LoadData()-metodin avulla. Käytännössä tämä tarkoitti tietokantayhteyden avaamista, tietokannan taulun sisällön lataamista SQLiteDataAdapteriin, DataTable:n luontia, sen täyttämistä SQLiteDataAdapterin avulla ja linkittämistä BindingSourceen. Tämän jälkeen vielä BindingSource asetettiin DataGridView:n DataSourceksi. Myös tiedon lisääminen toimi aluksi hitaasti, koska LoadData()-metodia kutsuttiin lisäämisen jälkeen. Tämän vuoksi uusi sanapari päätettiin lisätä DataGridView-kontrolliin uutena rivinä. Tässä yhteydessä haettiin lisätyn sanaparin rowid-arvo kannasta getRowId-metodin avulla, minkä seurauksena sanaparien lisääminen nopeutui käytettävyyden kannalta hyvälle tasolle.

Hakunopeus kasvoi huomattavasi, kun haussa alettiin käyttää DataView:n RowFilter-ominaisuutta. RowFilteria käyttämällä päästiin tilaan, jossa LoadData()-metodia jouduttiin kutsumaan ainoastaan sovelluksen käynnistyksen yhteydessä. RowFilterin avulla voidaan määritellä, mitä kaikkea DataGridView-kontrollissa näytetään. (Microsoft Developer Network a.) RowFilterissä on myös ominaisuus, joka tarjoaa käyttäjälle tarkempia hakuvaihtoehtoja. Esimerkiksi kirjoittamalla hakuriviin %sana%, näytetään käyttäjälle kaikki termit, jotka sisältävät %-merkkien sisään jäävän termin. Ominaisuus vaati erikoismerkkien käsittelyä ohjelmakoodissa, jotta sovellus ei kaatuisi tiettyjen merkkien syötön yhteydessä. Ennakoiva haku toimi RowFilterin käyttöönotonkin jälkeen hieman tökkien, mitä voidaan pitää käytettävyysongelmana. Ongelman ratkaisu vaatisi muutoksia käyttöliittymässä ja siinä, mitä kaikkea tietoa käyttäjälle näytetään ennakoivassa haussa. Haku hidastuu käsittääkseni sen takia, että ennakoivassa haussa päivitetään koko DataGridView. Jos päivitettävää osiota pienennettäisiin esimerkiksi tekemällä ennakoivalle haulle oma pieni ikkuna, voisi hakunopeus parantua.

Sovelluksessa on joitakin käyttöliittymällisiä ongelmia. Käyttäjä ei esimerkiksi pysty antamaan samalle sanaparille monta toimialaa. Nykyinen tietokantaratkaisu kyllä mahdollistaisi monen toimialan liittämisen yhdelle sanaparille, mutta ongelma on, miten näyttää tieto visuaalisesti käyttäjälle DataGridView-kontrollissa, ja miten käyttäjä voisi muokata tietoa. Tällä hetkellä toimialasarake on ComboBoxColumn, joka toisaalta näyttää yhden kohteen kerrallaan, mutta aktivoitaessa antaa käyttäjän näkyviin toimialojen pudotusvalikon. Yksi vaihtoehto olisi ComboBoxColumnin muokkaaminen siten, että se voisi näyttää samaan aikaan useamman kuin yhden kohteen. Käyttöliittymäongelmana voidaan myös pitää varsinaisen help-valikon puutetta. Käyttöliittymä sinällään on yksinkertainen, mutta mikäli käyttäjä ei muista, että pikanäppäinyhdistelmät saadaan näkyviin hiiren oikealla painikkeella, voi hän juuttua paikoilleen.

4.2 Käytettyjen työkalujen ja menetelmien arviointi

Visual Studio osoittautui vakaaksi ja helppokäyttöiseksi kehitysympäristöksi. Erityisesti verrattuna aikaisemmin opinnoissa käytettyyn Eclipse-kehitysympäristöön tuntui Visual

Studio loistavalta. Sovelluksen debuggaaminen oli suoraviivaista, ja Visual Studion IntelliSense-ominaisuus nopeutti koodaamista ja dokumentointia. Pidin erityisesti Visual Studion melko pitkälle viedystä mahdollisuudesta käyttää suunnittelunäkymää hyväksi sovelluksen kehittämisessä, jolloin muutokset näkyivät konkreettisina.

Jälkikäteen ajateltuna olisi voinut olla perusteltua valita WPF ja XAML Windows Formsin sijaan. WPF olisi antanut enemmän kustomointimahdollisuuksia käyttöliittymälle. Toisaalta Microsoftin tarjoamat laajat oppimis- ja tukiresurssit Windows Formsille sekä Windows Formsin pitkäaikainen vakiintunut asema API-tekniikkana mahdollistivat nopean oppimisen. Windows Formsin käyttöä helpotti Visual Studion graafinen suunnittelunäkymä, jolla käyttöliittymää voitiin kehittää visuaalisesti. Voi olla, että WPF:n oppimiseen olisi täytynyt panostaa enemmän aikaa kuin Windows Formsin opetteluun. Loppujen lopuksi ylimääräistä aikaa ei projektissa jäänyt.

C# suoritti ohjelmointikielenä tehtävänsä hyvin. Java-pohjalta siirtyminen C#-kieleen ei ollut iso hyppäys.

SQLite-tietokanta oli uusi ja positiivinen tuttavuus. Oletin, että SQLite olisi vaatinut edes jonkinnäköisen asennuksen, mutta käytännössä se ei vaatinut asennusta ollenkaan. Toimiakseen sovellus tarvitsi kansion, jossa Visual Studiolla tuotettu .exe-tiedosto ja System.Data.SQLite.dll-tiedosto sijaitsevat. Tämän lisäksi käyttäjä tarvitsee .NET Framework 4.5:n, joka asennetaan automaattisesti käyttöjärjestelmän puolesta, jos sitä ei jo entuudestaan käyttäjällä ole. System.Data.SQLite.dll täytyi kuitenkin etukäteen ladata, mutta se siirtyi käyttäjän koneelle mukavasti kansion siirtämisen yhteydessä. Mielestäni SQLite toimi nopeasti, mutta tulevaisuudessa esimerkiksi jatkokehityksen osalta kannattaisi varmasti testata muitakin tietokantaratkaisuja ja niiden nopeutta. En usko, että nykyisessä käyttötarkoituksessaan tietokantojen koot tulevat aiheuttamaan isoja ongelmia, mutta on mahdollista, että sovelluksen käyttö hidastuisi, kun SQLite-tietokanta kasvaisi suureksi.

4.3 Käytettävyystestaus

Sovelluksen käytettävyyttä testattiin projektin aikana useaan otteeseen. Kun tärkeä sovelluksen ominaisuus saatiin toteutettua, toimeksiantajalle selostettiin ja näytettiin, miten ominaisuutta käytetään sovelluksessa, ja toimeksiantaja sai sovelluksen uuden version kokeiltavakseen. Toimeksiantaja antoi ominaisuuksista palautetta, minkä perusteella sovellusta voitiin muuttaa. Ennakoivan haun muuttaminen vaihtoehtoiseksi oli yksi tällainen muutos.

Kun tärkeimmät ominaisuudet oli toteutettu, suoritettiin koko sovelluksen kattava käytettävyystestaus (liite 2). Käytettävyystestauksessa käyttäjälle annettiin tehtäviä, jotka hänen tuli suorittaa ilman apua. Tehtävät koskivat sovelluksen tyypillisiä käyttötapauksia. Testi suoritettiin toimeksiantajan tietokoneella hänen normaalissa työympäristössään. Testissä huomiota kiinnitettiin tehtävien suorituksen aikana tehtyihin virheisiin, niiden tasoon ja niistä toipumiseen. Käyttäjää pyydettiin tehtäviä suorittaessaan ajattelemaan ääneen, jotta testin seuraaja tietäisi käyttäjän suorittamien toimintojen taustalla olevat ajatukset ja mihin käyttäjän huomio kiinnittyy.

Suurin muutos käytettävyystestin perusteella oli oman ikkunan lisääminen toimialojen hallintaa varten.

4.4 Projektin eteneminen ja lopputulos

Projekti alkoi sovelluksen vaatimusten läpikäymisellä toimeksiantajan kanssa. Toimeksiantaja kertoi, mitä hän haluaa sovelluksella voitavan tehdä. Näistä koottiin vaatimukset, jotka toimeksiantaja priorisoi niiden tärkeyden perusteella. Osa vaatimuksista oli selkeitä ja ymmärrettäviä heti alusta alkaen, mutta osa selventyi vasta projektin kuluessa. Vaatimukset olivat lähinnä toiminnallisia ja sovelluksen toteutukseen annettiin lähestulkoon vapaat kädet.

Vaatimuksissa ei esimerkiksi ollut toimialojenhallintaa, mutta sovelluksen ylläpidettävyyden kannalta oli selvää, että toimialoja täytyisi voida muokata ja poista, ei ainoastaan lisätä. Periaatteessa käyttäjällä on jo mahdollisuus itse muokata tietokoneellaan sijaitsevan SQLite-tietokannan sisältöä miten haluaa, mutta tämä ei

luonnollisesti ole järkevä ratkaisu, koska käyttäjällä ei ole tarvittavia teknistä taitoa ja tietoa tietokannan hallinnointiin. Sovellukseen lisättiin erillinen ikkuna, josta toimialoja voidaan lisätä, muokata ja poistaa.

Kuten jo aikaisemmin mainittiin, käyttäjä halusi, että sovellusta voitaisiin käyttää pikanäppäimillä, koska hän ei pidä hiiren käytöstä. Sovellus kehitettiinkin niin, että sitä voidaan käyttää lähes täysin ilman hiirtä. Aluksi ainoastaan tärkeimmät ominaisuudet, kuten sanojen haku, lisäys ja poisto, toteutettiin pikanäppäimillä toimiviksi, mutta pikanäppäimillä toimimista laajennettiin myöhemmin koko ohjelman käyttöliittymään. Sovelluksessa voidaan navigoida tabulaattorin ja nuolinäppäimien avulla. Käyttäjä pystyy myös vaihtamaan hakutyyppejä ja kielikohtaisen haun suuntaa pikanäppäimillä. Näin tekemällä saavutettiin käyttäjän kannalta mukava ratkaisu. Toisaalta suurempi pikanäppäinten määrä vaatii käyttäjältä enemmän opettelua ja muistamista, mutta kun käyttäjä hallitsee pikanäppäimet, nopeutuu sovelluksen käyttö varmasti, mikä antaa positiivisen käyttökokemuksen ja edistää käytettävyyttä.

Koska Windows Forms ja SQLite-tietokanta olivat lähes täysin tuntemattomia, eikä C#-kielestä ja Visual Studiostakaan ollut enempää kokemusta kuin yhden pintaa raapaisevan kurssin verran, kului paljon aikaa teknologioiden tutkimiseen ja asioiden selvittämiseen. Projektissa saatiin aikaan vaatimuksia vastaava sovellus lähes tuntemattomilla työkaluilla ja tekniikoilla. Siksi olen lopputulokseen tyytyväinen.

4.5 Oppiminen

Projekti oli hyvin opettava kokemus niin sovelluksen kehittämisen kuin projektityön kannalta. Visual Studiota, C#-kieltä ja .NET Frameworkia oli käytetty ainoastaan yhden kurssin aikana, mutta pidin kehitysympäristöstä niin paljon, että minua kiinnosti alkaa rakentaa sovellusta juuri tässä kehitysympäristössä. Oman osaamisen kannalta olisi ollut luontevampaa valita ohjelmointikieleksi Java ja kehitysympäristöksi Eclipse, koska näihin teknologioihin opinnot olivat hyvin pitkälti painottuneet. C#-kieleen pääsi hyvin sisälle Java-pohjan kautta, kun taas Visual Studio oli todella positiivinen uusi tuttavuus Eclipseen verrattuna.

Työpöytäsovelluksia ei ollut aikaisemmin käsitelty opinnoissa lainkaan, joten työpöytäsovelluksen kehittäminen oli myös uusi ja kiinnostava kokemus. Opin myös paljon Windows Formsista ja sen sisältämien kontrollien käsittelystä, toiminnasta sekä ihan käytännössä siitä, miten Windows Forms -sovellus toteutetaan. Aikaisempaa varsinaista kokemusta Windows Formsista ei itselläni ollut, muuta kuin että olin tutkinut sitä omatoimisesti jonkin verran.

Projektityö oli toki ollut suuri osa ohjelmistokehityslinjan koulutusta, mutta kurssien aikana projektityö oli jakautunut hyvin epätasaisesti, jolloin siitä ei saanut samanlaista kuvaa kuin nyt, kun koko projekti toteutettiin itsenäisesti alusta loppuun. Juuri projektityöosuus oli erittäin opettava kokemus.

5 Yhteenveto

Työn lopputulos vastasi vaatimuksia (Liite 1) hyvin. Työn lopputuloksena syntyi toimiva sovellus, joka on kielenkääntäjälle hyödyllinen apuväline. Kaikki halutut toiminnalliset vaatimukset saatiin toteutettua, paitsi vaatimus ulkopuolisten sanastojen tuomisesta ohjelmaan. Kyseinen ominaisuus oli matalimman mahdollisen prioriteetin vaatimus.

Ulkopuoliset sanastot voidaan jakaa ulkopuolisiin, usein kaupallisiin sanastoihin, jotka eivät siis ole kääntäjän itse tuottamia, ja ulkopuolisiin ei-kaupallisiin sanastoihin, jotka kääntäjä on itse aikaisemmin rakentanut edellisten käännöstöiden yhteydessä. Jälkimmäiset ovat pelkkiä Word-dokumentteja. Vaatimuksen voidaan kuitenkin katsoa osittain toteutuneen, koska ohjelmaan tuotiin ohjelmallisesti Databexit Oy:n omistama 117 000 sanaparin sanasto, jota myös käytettiin ohjelman testauksessa. Olisi ihanteellista, jos käyttäjä pystyisi itse tuomaan asiakaskohtaisia, jo olemassa olevia sanastojaan suoraan ohjelmaan esimerkiksi Word- tai Excel-dokumenteista. Ulkopuolisten kaupallisten sanastojen tuominen ohjelmaan ei ollut tässä projektissa ajankohtaista.

5.1 Ajatuksia jatkokehityksestä

Projektin aikana jatkokehitysehdotuksia tuli sekä toimeksiantajalta että omasta puolestani. Sovellusta voitaisiin viedä siihen suuntaan, että kaikkiin sovelluksen perusominaisuuksiin, kuten esimerkiksi sanojen lisäämiseen, päästäisiin käsiksi suoraan DataGridView-kontrollin kautta. Sanaparien muokkaaminen ja poistaminen tapahtuu jo nyt suoraan DataGridView-kontrollissa, joten voisi olla järkevää toteuttaa sanojen lisääminen samasta paikasta. Nyt sanojen ja toimialan lisäämiseen avataan oma dialogi. Käyttäjälle voitaisiin myös antaa enemmän vapauksia muokata ohjelman käyttöliittymää halunsa mukaan. Hyödyllisiä ominaisuuksia voisi olla sovelluksen fonttien, kirjasinkokojen ja värien muokkaus. Voisi myös olla järkevää antaa käyttäjälle mahdollisuus määrittää pikanäppäimet itse. Näitä ominaisuuksia varten voitaisiin tehdä oma valikko.

Käyttäjälle olisi varmasti hyödyllistä, jos hän pystyisi tuomaan sovelluksen käyttöön jo valmiita, itse rakennettuja sanastoja, joita hän on käyttänyt aikaisempien käännöstöiden yhteydessä. Toki käyttäjä voi manuaalisesti kopioi-liimaa-menetelmällä lisätä sanaparit yksi kerrallaan, mutta se ei ole mielekästä, jos sanastot ovat suuria ja niitä on paljon. Oikeastaan tämän voidaan katsoa kuuluvan edellä mainittuun vaatimukseen ulkopuolisten ei-kaupallisten sanastojen tuomisesta sovellukseen.

Ennakoivan haun nopeutta voitaisiin tehostaa esimerkiksi siten, että lisättäisiin oma pieni kontrolli tai ikkuna vaikkapa sovelluksen yhteen nurkkaan, jossa näytettäisiin haut ennakoivasti yhden kielen perusteella. Tällöin koko DataGridView-kontrollia ei tarvitsisi päivittää ennakoivassa haussa, vaan uuden pienen kontrollin päivittäminen riittäisi. Näin tekemällä ennakoiva haku luultavasti tuntuisi nopeammalta hakua kirjoitettaessa, koska enterillä toimiva ei-ennakoiva haku toimii lähes välittömästi. Esimerkiksi MOT:n elektroninen sanakirja käyttää tämänkaltaista ratkaisua, mutta varsinainen ennakoiva haku ei MOT:n ratkaisussakaan toimi nopeammin – erona on se, että hakurivi päivittyy käyttäjän sitä muuttaessa nopeasti, jolloin hän kokee haun olevan nopea. Se, kuinka nopeasti hakuriville muodostuu tekstiä, ei sinällään kerro itse hakunopeudesta mitään.

Käyttäjälle olisi varmasti hyvä antaa mahdollisuuksia tarkempiin hakuihin. Uudenlaisista hakutyypeistä olisi ehdottomasti hyötyä, jos sovelluksen sisältämät sanastot kasvavat suuriksi. Sovelluksen testikäytössä olleen 117 000 sanaparin sanaston hauissa ilmenee jo omasta mielestäni ja myös käyttäjän mielestä selvästi tarve tarkemmille hakuvaihtoehdoille.

Voi olla järkevää pohtia, kuvaako sana 'toimiala' parhaiten sitä, mitä käyttäjä haluaa termillä kuvattavan. Projektin aikana kävi ilmi, että toimiala voi usein myös tarkoittaa asiakasta. Käyttäjä on tottunut kategorisoimaan töitään asiakkaan perusteella. Tällä hetkellä sovelluksessa ei ole tehty varsinaisesti mitään erotusta toimialan ja asiakkaan välille, vaan toimiala voi pitää sisällä asiakkaan, mikäli käyttäjä niin päättää. Esimerkiksi uuden käännöksen yhteydessä kääntäjä saattaa päättää alkaa kategorisoida uutta sanastoa työn nimen tai asiakkaan perusteella. Toimiala ja asiakas ovat kääntäjän

kannalta siis sama asia – on oikeastaan vain kyse semantiikasta ja siitä, miten esimerkiksi toimialakohtainen haku halutaan käyttäjälle esittää. Onko oikein puhua toimialasta, vai pitäisikö käyttää jotain sanaa, joka kuvaisi haluttua kohdetta tarkemmin?

Varteenotettava vaihtoehto on myös toteuttaa erillinen kenttä asiakkaalle ja/tai työn nimelle, jolloin toimiala olisi yksiselitteisesti ymmärrettävissä. Uusien kenttien lisäämisessä olisi se hyvä puoli, että käyttäjä pystyisi hakemaan ja kategorisoimaan sanoja entistä tarkemmin, mutta jos kyseisten kenttien lisääminen ei ole välttämätöntä, on mahdollista, että käyttäjä ei hyödyntäisi kenttiä. Pahimmassa tapauksessa kentät siis jäisivät tyhjäksi, eivätkä auttaisi käyttäjää millään tavalla. Ne voisivat jopa heikentää sovelluksen käyttökokemusta, koska tyhjät kentät vievät turhaa tilaa sovelluksessa.

Mahdollisuutta ja tarvetta monen hakutuloksen näyttämiselle samaan aikaan voitaisiin pohtia. Esimerkiksi kahden haun erottaminen toisistaan välilehdillä voisi olla hyödyllinen ominaisuus. Hakuehdot voisivat myös tallentua sovelluksen muistiin, jolloin sovellus aukeaisi samoilla asetuksilla ja hauilla kuin se suljettiin. Tällöin käyttäjän ei tarvitsisi käännöstyön aikana aina erikseen määrittää haluamansa asetuksia sovelluksen käynnistyksen yhteydessä, vaan ne olisivat aina valmiina.

Lähteet

Chapple, M. Should I Normalize My Database?. Luettavissa: http://databases.about.com/od/specificproducts/a/Should-I-Normalize-My-Database.htm. Luettu: 30.12.2013

Connolly, R. 2007. Core Internet Application Development with ASP.NET 2.0. Pearson Education. New Jersey.

ECMA International 2006. ECMA-334 Standard: C# Language Specification. ECMA International. Geneve. Luettavissa: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf. Luettu: 30.12.2013

Elliott, M. 2010. C# and XML Source Code Documentation. Luettavissa: http://www.codeproject.com/Articles/11082/C-and-XML-Source-Code-Documentation. Luettu: 30.12.2013

Friedl, S. 2007. SQL Injection Attacks by Example. Luettavissa: http://www.unixwiz.net/techtips/sql-injection.html. Luettu: 30.12.2013

Janssen, C. Iterative and Incremental Development. Luettavissa: http://www.techopedia.com/definition/25895/iterative-and-incremental-development. Luettu: 30.12.2013

Microsoft Developer Network a. DataView.RowFilter Property. Luettavissa: http://msdn.microsoft.com/en-us/library/system.data.dataview.rowfilter%28v=vs.110%29.aspx. Luettu: 30.12.2013

Microsoft Developer Network b. Designing for Web or Desktop? Luettavissa: http://msdn.microsoft.com/en-us/library/ms973831.aspx. Luettu: 30.12.2013

Microsoft Developer Network c. .NET Framework 4.5. Luettavissa: http://msdn.microsoft.com/en-us/library/w0x726c2%28v=vs.110%29.aspx. Luettu: 30.12.2013

Microsoft Developer Network d. Properties Window. Luettavissa: http://msdn.microsoft.com/en-us/library/ms171352.aspx. Luettu: 30.12.2013

Microsoft Developer Network e. Toolbox. Luettavissa: http://msdn.microsoft.com/en-us/library/2381cd09.aspx. Luettu: 30.12.2013

Microsoft Developer Network f. Visual C# IntelliSense. Luettavissa: http://msdn.microsoft.com/en-us/library/43f44291.aspx. Luettu: 30.12.2013

Microsoft Developer Network g. Windows Forms Overview. Luettavissa: http://msdn.microsoft.com/en-us/library/8bxxy49h%28v=vs.110%29.aspx. Luettu: 30.12.2013

Microsoft Developer Network h. XML Documentation Comments (C# Programming Guide). Luettavissa: http://msdn.microsoft.com/en-us/library/b2s063f7.aspx. Luettu: 30.12.2013

Microsoft Technet. SQL Injection. Luettavissa: http://technet.microsoft.com/en-us/library/ms161953%28v=sql.105%29.aspx. Luettu: 30.12.2013

Nielsen, J. 1993. Usability Engineering. Academic Press Limited. London.

Sheriff, P. 2002. Designing a .NET Application. Luettavissa: http://msdn.microsoft.com/en-us/library/ms973829.aspx. Luettu: 30.12.2013

SQLite a. About SQLite. Luettavissa: http://www.sqlite.org/about.html. Luettu: 30.12.2013

SQLite b. Appropriate Uses For SQLite. Luettavissa: http://www.sqlite.org/whentouse.html. Luettu: 30.12.2013

SQLite c. Datatypes in SQLite. Luettavissa: http://www.sqlite.org/datatype3.html. Luettu: 30.12.2013

SQLite d. Distinctive Features of SQLite. Luettavissa: http://www.sqlite.org/different.html. Luettu: 30.12.2013

SQLite e. Limits in SQLite. Luettavissa: http://www.sqlite.org/limits.html. Luettu: 30.12.2013

SQLite f. SQLite is self-contained. Luettavissa: http://www.sqlite.org/selfcontained.html. Luettu: 30.12.2013

SQLite g. SQL As Understood By SQLite. Luettavissa: http://www.sqlite.org/lang_createtrigger.html. Luettu: 30.12.2013

SQLite Manager. Luettavissa: https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/. Luettu: 30.12.2013

Troelsen, A. 2008. Pro C# 2008 and the .NET 3.5 Platform Fourth Edition. Springer-Verlag. New York.

W3C 2001. XML in 10 points. World Wide Web Consortium (W3C). Luettavissa: http://www.w3.org/XML/1999/XML-in-10-points.html.en. Luettu: 30.12.2013

Liitteet

Liite 1. Tuoteominaisuudet.

	Toimijana	minä …	jotta ...	Prioriteetti	milloin tehdään	tilanne	kommetteja / avoimia asioita
1	Sanaston käyttäjä	haluan lisätä uusia sanoja sanastoon helposti ja nopeasti	sanastoa voi täydentää	1	1. seurantakokoukseen	tehty	
2	Sanaston käyttäjä	haluan, että sanasto aukeaa nopeasti ja helposti	ohjelmaa on mukava käyttää	2	2. seurantakokoukseen	tehty	
3	Sanaston käyttäjä	haluan, että sanat saa aakkosjärjestykseen helposti	haluamat sanat löytää kätevästi	2	2. seurantakokoukseen	tehty	
4	Sanaston käyttäjä	haluan, että sanat saa aakkosjärjestykseen sekä englannin että suomen mukaan, jolloin sarakkeet vaihtuisivat niin, että aakkosjärjestyksessä olevat sanat olisivat aina vasemmalla.	haluamat sanat löytää kätevästi	3	2. seurantakokoukseen	tehty	
5	Sanaston käyttäjä	haluan siirtää sanastoon alkutekstin ja kohdetekstin sanan esimerkiksi napauttama sitä hiirellä tai kopio-liimaamalla.	ei tarvitse itse kirjoittaa sanoja uudelleen		päättökokoukseen	tehty	
6	Sanaston käyttäjä	haluan, että sanastossa on alkutekstin sanan ja kohdetekstin sanan lisäksi olla kommentointimahdollisuus (vaikka esimerkkilause), asiakkaan / työn nimi ja päivämäärä, joka voisi kirjautua automaattisesti ja olla aina uusin, jos sanaa tai vastinetta on muutettu. Päivämäärän ei pakko olla aina näkyvillä, vaan sen voisi esim näpäyttää auki.	voin laittaa muistutuksia itseäni varten	4	2. seurantakokoukseen	tehty	
7	Sanaston käyttäjä	haluan, että samalle sanalle on mahdollista lisätä useita vastineita. Nämä voivat olla myös omilla hakuriveillään.		3	2. seurantakokoukseen	tehty	

8	Sanaston käyttäjä	haluan muokata vanhoja sanoja vastineineen.		3	2. seurantakokoukseen	tehty	
9	Sanaston käyttäjä	haluan, että sanastoon voisi tuoda kokonaisia ulkopuolisia sanastoja.	lähtökohta sanaston käyttöön olisi parempi		päättökokoukseen	ei valmis – sanastoon tuotiin Databexit oy:n omistama sanasto, mutta ulkopuolisia sanastoja ei.	
10	Sanaston käyttäjä	haluan sanojen hakuun hakurivin, joka olisi ennakoiva, eli ehdottaisi sanoja sitä mukaa, kun hakuriville kirjoittaa.	sanojen hakeminen olisi helpompaa		päättökokoukseen	tehty	sanat tulevat näkyviin datagridviewiin, jota päivitetään aina, kun käyttäjä muuttaa hakurivin sisältöä
11	Sanaston käyttäjä	haluan hakea sanoja erikoisalan, asiakkaan tai työn mukaan.		4	2. seurantakokoukseen	tehty	
12	Sanaston käyttäjä	haluan hakea sanoja pelkän päivämäärän mukaan ja näyttää ne kronologisessa järjestyksessä.			päättökokoukseen	tehty	

HAAGA-HELIA
ammattikorkeakoulu

Käytettävyystestaus

<Opinnäytetyö>

Markus Becks

Testattava sovellus

Käytettävyystestauksen kohteena on kielenkääntäjän sanastotyökalu. Sovelluksen toteutuksessa pyrittiin vähentämään käyttäjän hiiren käyttöä ja nopeuttamaan oikeiden termien ja sanojen käyttöä.

Testin päämäärä

Käytettävyystestin tarkoituksena on testata sovelluksen perustoimintoja.

Vastuut

Testin valvojana ja tarkkailijana toimi opinnäytetyön tekijä. Tarkkailijan rooli oli antaa testikäyttäjälle tehtäviä, ei auttaa tehtävien suorittamisessa. Mikäli testikäyttäjä olisi juuttunut johonkin kohtaan, olisi tarkkailija voinut auttaa testikäyttäjää pääsemään eteenpäin.

Testaaja

Testaajana toimi sovelluksen tuleva käyttäjä. Testaaja on itse määritellyt, mitä ominaisuuksia hän ohjelmaan haluaa. Hän tuntee ohjelman ja on kokeillut sitä pintapuolisesti.

Testin kulku

Testin aikana testikäyttäjälle annettiin tehtäviä, jotka tämän tuli suorittaa ilman apua. Käyttäjää pyydettiin ajattelemaan tehtävien aikaan ääneen. Tehtävän jälkeen käyttäjän antama palaute ja testitulokset kirjattiin ylös, minkä jälkeen siirryttiin seuraavaan tehtävään.

Tehtävät

Alla olevassa taulukossa on esitetty ja numeroitu tehtävät, jotka käyttäjää pyydettiin suorittamaan.

Nro.	Tehtävä
1	Etsi suomenkielinen sana ”ominaisuus”
2	Lisää sanastoon sanapari ”disorganization – epäjärjestelmällisyys”
3	Muokkaa sanaparia ”test word – testisana” siten, että poistat suomenkielisen vastineen sanaparista
4	Poista sanapari ”a little bit – hieman”
5	Hae sanastosta kommentin ”asiakas Kalle” perusteella
6	Hae päivämäärän 7.12. perusteella
7	Lisää toimiala ”satunnainen”

Tulokset

Alla olevassa taulukossa on esitetty testin tulokset.

Tehtävänumero	Virheet	Kommentit
1	Ei virheitä	Käyttäjä muisti CTRL+F pikanäppäimen käytön haun yhteydessä, eikä aktivoinut hakukenttää kursorilla.
2	Ei virheitä	Käyttäjä avasi sanaparien lisäyslomakkeen ja lisäsi sanaparin oikein. Käyttäjä huomautti, ettei sanapari tullut näkyviin heti lisäyksen jälkeen.
3	Pieni virhe. Käyttäjä painoi ennakoivassa haussa enteriä, mikä ei ole tarpeen. Virheellä ei ollut vaikutusta toiminnan kannalta.	Käyttäjä löysi sanaparin, mutta mietti hetken miten yksittäisen sanan poisto tapahtuu. Hän poisti sanan DataGridView:stä delete-näppäimellä.
4	Pieni virhe. Käyttäjä poisti ensin vain yhden sanan	Käyttäjä poisti vain yhden sanan deletellä. Hän mietti hetken, klikkasi hiiren oikeaa painiketta ja

	käyttämällä delete-näppäintä. Virheellä oli ainoastaan aikaa vievä vaikutus.	huomasi, että CTRL+D poistaa koko sanaparin, minkä hän myös tämän jälkeen teki.
5	Ei virheitä	Käyttäjä mietti tovin, ja suoritti haun. Käyttäjä huomautti, että laaja haku on kätevä, koska sillä voi etsiä myös kommenttikentästä.
6	Virhe. Käyttäjä ei muistanut, että laajalla haulla voidaan hakea myös päivämääriä.	Käyttäjä etsi ensin päivämäärää manuaalisesti leima-sarakkeesta, mutta löysi kyllä haluamansa tuloksen. Tämän jälkeen käyttäjälle muistutettiin, että päivämääriä voi hakea laajan haun kautta. Käyttäjä onnistui lopulta haussa.
7	Suuri virhe. Käyttäjä muutti huomaamattaan sanaparin toimialaa.	Käyttäjä pohti, miten toimiala voidaan lisätä. Hän navigoi DataGridView:ssä toimiala-sarakkeeseen ja alkoi kirjoittaa erään sanaparin toimialasolun päälle, jolloin sanaparin toimiala muuttui.

Tulosten tulkinta ja kehitysehdotukset

Testitulokset olivat rohkaisevia, mutta jotkut osa-alueet kaipasivat selkeästi muutosta. Perustoiminnot sujuvat käyttäjältä hyvin, vaikka välillä käyttäjällä oli ongelmia muistaa näppäinyhdistelmiä, esimerkiksi tehtävässä 4. CTRL+D on näppäinyhdistelmänä looginen valinta Deletelle, joten uskon, että ajan myötä tilanne korjaantuu.

Tehtävässä 6 käyttäjä ei muistanut, että päivämääriä voidaan hakea laajalla haulla. Tästä johtuen ehdotettiin tooltipin lisäämistä laajan haun yhteyteen, jotta käyttäjä voi halutessaan tarkistaa, mitä laaja haku teki. Tämäkin vaihtoehto on parempi, kuin ettei laajan haun toimintaa olisi selostettu millään tavalla. Voisi olla järkevää dokumentoida käyttäjälle hakujen toimivuudet ohjelmassa esimerkiksi help-valikkona.

Tehtävä 7:n tulos oli jossakin mielessä odotettu, koska toimialojen hallintaa ei ollut toteutettu. En kuitenkaan olettanut, että käyttäjä olisi voinut jopa muuttaa toisten sanaparien toimialoja etsiessään keinoa lisätä toimiala. Toimialan olisi voinut lisätä sanaparien lisäys -ikkunassa. Sanakentät olisi jätetty tyhjiksi, ja toimiala-pudotusvalikkoon olisi kirjoitettu halutun toimialan nimi. Tämä ei

missään nimessä ole järkevä ratkaisu, joten päätimme toteuttaa oman ikkunan toimialojen hallinnalle.