VBA-OHJELMOINNILLA TEHOA JA LAATUA MS EXCELILLÄ SUORITETTAVIIN TEHTÄVIIN

CASE KPI-TYÖKALU

Janne Satola

Opinnäytetyö
Toukokuu 2012
Tietojenkäsittelyn koulutusohjelma

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

SATOLA, JANNE:
VBA-ohjelmoinnilla tehoa ja laatua MS Excelillä suoritettaviin tehtäviin
Case KPI-työkalu

Opinnäytetyö 42 sivua, josta liitteitä 1 sivu
Toukokuu 2012

Opinnäytetyö on kaksiosainen ja koostuu tutkimusraportista sekä erillisestä Visual Basic for Applications -ohjelmoinnilla (VBA) toteutetusta ohjelmasta, joka kehitettiin globaalisti teollisuuden alalla toimivalle toimeksiantajalle.

Raportissa on kuvattu yksinkertaisesti ja käytännönläheisesti millaisia Excel-taulukkolaskentaohjelmalla suoritettavia toimintoja on mahdollista automatisoida VBA-ohjelmoinnilla. Koodin tuottamisen perusteet esitellään lyhyesti, samoin kuin VBA-ohjelmoinnissa yleisimmin hyödynnettävät ominaisuudet.

Toimeksiantaja halusi tehostaa KPI-seurantaan (Key Performance Indicators) kuukausittain käytettävää työaikaa, johon kului toimeksiannon aikaan yhteensä 1,5-2,5 työpäivää kahdelta henkilöltä. Työn tehostamiseksi toteutettiin opinnäytetyönä VBA-ohjelma, joka suorittaa automaattisesti Excelillä ne tehtävät, jotka toimeksiantajan työntekijät tekivät aiemmin käsin.

Työkalun edut ovat kiistattomat. Ohjelma sekä tarkastaa 51:n seurannassa käytettävän lähdeaineiston sisällön että suorittaa näille tarvittavat muokkaustoiminnot 4 minuutissa ja 10 sekunnissa. Yhtä aineistoa kohden toimintoihin kuluu 5,2 sekuntia johon sisältyy myös tarvittavan tiedoston avaaminen. Aikaa koko KPI-seurantaprosessiin kuluu työkalua käyttäen hieman alle 4 tuntia. Tämän ajan määrä ei ole vakio, sillä työhön liittyy edelleen ihmisen tekemästä työstä aiheutuvaa vaihtelua.

Työkalu täyttää kaikki toimeksiantajan sille asettamat laatuvaatimukset. Tästä huolimatta se tarvitsee vielä jatkokehitystä, jotta mahdollisimman korkea käyttäjäystävällisyys saavutetaan. Työkaluun tulee kehittää erillinen apuohjelma, jonka avulla ylläpitäjä voi selvittää, miltä lähdeaineistolta hän voi kopioida tarvittavat ylläpidot työkalun käsiteltäväksi lisättävälle uudelle lähdeaineistolle. Lopullinen tavoite on tehdä työkalu tarpeettomaksi siirtämällä kaikki tämän sisältämät toiminnallisuudet toimeksiantajan varsinaiseen KPI-seurantajärjestelmään.

Asiasanat: ohjelmointi, Visual Basic for Applications, prosessien tehostaminen

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree of Business Information Systems

SATOLA, JANNE:
VBA Programming – Gaining Efficiency and Quality for Advanced MS Excel Tasks
Case KPI Tool

Bachelor's thesis 42 pages, appendices 1 page
May 2012

The thesis work consists of two different parts. The first part is this report and the second one is a separate programme, which was developed by using the Visual Basic for Applications programming language (VBA). The programme was developed for a client company operating globally in the field of industrial business.

This report describes briefly and practically the kinds of Excel functions which are possible to automate with VBA programming. Additionally, the production of the code used is introduced briefly as well as the most commonly used features in VBA programming.

The client had the need to improve the processes related to their KPI follow-up (Key Performance Indicators). Before this project, the client had used 1,5 – 2,5 working days per month for this reporting performed by two people. To improve this process, a VBA programme was implemented which accomplishes automatically all of these tasks previously performed by employees.

The advantages of the programme are unquestionable. The programme is able to ensure that all the required 51 source files contain error-free source data and the programme also runs the required editing tasks for these data. This whole process takes only 4 minutes and 10 seconds. It makes only 5,2 seconds per one source data and this also includes the time that it takes to open a required source file. Nowadays the whole KPI follow-up process takes a little less than four hours. The duration is not always the same, as there are still some manual tasks which cause some variation.

The KPI tool meets all the quality requirements defined by client. Despite this, it still needs some further development so that highest possible user friendliness will be achieved. There is a need for a separate utility program, which tells to admin from which existing source data they may copy the correct settings to the source data which can then be added as a part of the tool's processes. The final goal is to make the developed tool useless by transferring all its functions to client's proper KPI follow-up system.

Keywords: programming, Visual Basic for Applications, improving the processes

SISÄLTÖ

LYHENTEET JA TERMIT

Doorman	Opinnäytetyönä kehitetty VBA-ohjelma
Funktio	Aliohjelma, joka pystyy palauttamaan isäntäohjelmaan dataa muuttujan avulla
KPI	Key Performance Indicators
Makro	VBA-ohjelma, jonka avulla suoritetaan käyttäjän puolesta työkirjalle tehtävät toiminnallisuudet automaattisesti
MIS	Management Information System
Muuttuja	Ohjelmoinnissa käytettävä tietovarasto, johon on mahdollista väliaikaisesti tallentaa informaatiota ja josta sitä on myös mahdollista hakea
Peruskäyttäjä	Käyttäjä, joka käyttää ohjelmaa ainoastaan siinä mittakaavassa, johon ohjelma on suunniteltu, ja hyödyntää ainoastaan ohjelman vakiotoimintoja
Proseduuri	Aliohjelma, joka ei pysty palauttamaan dataa tämän isäntäohjelmaan
Taulukko	Excel-työkirjan välilehti
Työkirja	Excel-tiedosto, joka sisältää yhden tai useampia taulukoita
VBA	Visual Basic for Applications
VBA-editori	VBA-ohjelmien kirjoittamiseen ja muokkaamiseen tarkoitettu työkalu
VBA-moduuli	Makrojen tallentamiseen tarkoitettuja säiliöitä
Virus	Haittaohjelma
Ylläpitäjä	Ohjelman toimivuudesta vastaava henkilö, joka vastaa ohjelman toimintaan liittyvien ylläpitojen suorittamisesta.

1 JOHDANTO

1.1 Tausta

Olen tottunut automatisoimaan kaikki Excelillä säännöllisesti tehtävät toiminnot, joiden suorittamisesta on aiheutunut työnantajilleni turhia palkkakuluja. En arvostele tehdyn työn lopputuloksena syntyvien raporttien tarvetta yrityksille, ainoastaan näiden raporttien koostamiseen käytetyn työajan kestoa. Tätä arvostelen siksi, että haluttuun lopputulokseen päästäisiin tehokkaammin automatisoimalla VBA-ohjelmilla tarvittavat Excel-toiminnot. Nämä ohjelmat, joita arkisemmin kutsutaan makroiksi, suoriutuvat tehtävistä murto-osassa siitä ajasta, joka ihmisellä menee saman työn suorittamiseen.

Ajankäytön tehostumisen lisäksi käyttäjän tekemät virheet poistuvat, mikäli VBA-ohjelma on kirjoitettu oikeellisesti ja huomioiden kaikki muuttuvat tekijät. Automatisoinnin seurauksena työtehtävien mielekkyys kasvaa yksitoikkoisten töiden vähentyessä ja tätä kautta myös työssä viihtyminen paranee.

VBA-ohjelmointia tulisi hyödyntää apuna myös harvemmin suoritettavissa tehtävissä. Mikäli tehtävä vie paljon aikaa, on perusteltua kirjoittaa tätä varten oma ohjelma, vaikka työ suoritettaisiin ainoastaan kertaalleen. Mitä pitempään työntekijä tekee samaa yksitoikkoista työtä, esimerkiksi kahden eri tiedoston välillä tehtävää tietojen kopiointia, virheriski kasvaa jatkuvasti työn edetessä ja tekijän väsyessä.

1.2 Tavoite ja tarkoitus

Opinnäytetyön tavoitteena on tehdä VBA-ohjelmointia tunnetuksi ja kertoa tämän ohjelmointitavan tarjoamista mahdollisuuksista. On outoa, että vain harvoissa yrityksissä hyödynnetään VBA-ohjelmia, vaikka näiden avulla olisi mahdollista suorittaa tehokkaammin kaikki samat toiminnot jotka Excelin käyttäjät tekevät käsin. Tämän takia tavoitteena on myös, että opinnäytetyön avulla esimies- ja asiantuntijatehtävissä olevat henkilöt osaavat sekä tunnistaa automatisoitavat tehtävät että hakea tarvittavan ohjelman kehittämiseen apua esimerkiksi edustamansa yrityksen IT-osaston kautta.

Toisena tavoitteena on kertoa Office-tuoteperheeseen liittyvistä tietoturvariskeistä, sillä vain harvat tietokoneen peruskäyttäjät ovat näistä tietoisia. Tarkoituksena on kertoa, että uhkia on olemassa, vaikkei tietokoneen käyttäjä itse käyttäisi lainkaan VBA-ohjelmia.

Opinnäytetyön teknisen osuuden tarkoituksena on tehostaa globaalisti teollisuuden alalla toimivan toimeksiantajan KPI-seurantaprosessia (Key Performance Indicators). KPI-seurannalla tarkoitetaan keskeisten suorituskykyä kuvaavien tunnuslukujen jatkuvaa seurantaa sekä näiden lukujen pohjalta tehtävien päätösten tekoa. Toimeksiantaja ei halunnut yrityksen nimeä julkaistavan opinnäytetyössä, joten yrityksestä käytetään raportilla ainoastaan nimitystä toimeksiantaja. Tehostumisen on tarkoitus tapahtua Excel-työkalun avulla, joka suorittaa jatkossa työvaiheet, jotka aiemmin on suoritettu Excelillä toimeksiantajan työntekijöiden toimesta. Tehostamista tapahtuu, mikäli aiemmin tehtäviä hoitaneet henkilöt onnistutaan vapauttamaan Excelin ylläpitotyöstä, mahdollistaen heidän keskittymisen todellista asiantuntijuutta vaativiin tehtäviin.

1.3 Tutkimuskysymykset

Ensimmäinen kysymys, johon opinnäytetyössä haetaan vastausta, on: ”Mitkä toimeksiantajan KPI-seurantaan liittyvät prosessit on mahdollista automatisoida VBA-ohjelmoinnin avulla?”. Toinen tutkimuskysymys on: ”Paljonko VBA-ohjelma säästää kuukausittain työaikaa verrattuna aiempaan työtapaan?”. Kolmas tutkimuskysymys on: ”Onko KPI-työkalusta mahdollista kehittää riittävän skaalautuva, eli onko VBA-ohjelmointia taitamattoman henkilön mahdollista sekä käyttää että ylläpitää työkalua pelkkien kirjallisten ohjeiden avulla?”.

1.4 Rajaus

VBA:n hyödyntämistä käsitellään ainoastaan Excelissä, vaikka tämän ohjelmointikielen käyttö on mahdollista kaikissa muissakin Office-tuoteperheen ohjelmissa. Opinnäytetyö ei ole VBA:n käyttö- eikä käyttöönotto-opas, ainoastaan tietoisku tämän ohjelmointikielen tarjoamista mahdollisuuksista sekä yleisimmin käytetyistä toiminnallisuuksista.

Kehitettävä KPI-työkalu käsittelee toimeksiantajan luottamuksellista dataa, jonka sisältö ei ole merkityksellistä opinnäytetyön kannalta. Tämän takia kyseistä dataa ei julkaista. Toimeksiantajan työtehokkuudessa tapahtunutta muutosta seurataan mittaamalla työn suorittamiseen kuluva aika sekä ennen ohjelman käyttöönottoa että tämän jälkeen.

Toimeksiantajan varsinainen KPI-seurantajärjestelmä on jatkossa kolmannen osapuolen toteuttama Advanced Excel -ohjelma. Tätä järjestelmää ei käsitellä raportissa muutamia viittauksia lukuun ottamatta, sillä tämän järjestelmän toiminnallisuudet eivät suoranaisesti liity opinnäytetyöhön.

1.5 Menetelmät ja aineisto

Tutkimusmenetelmänä käytettiin toimintatutkimusta, jossa yhdistyy tutkimuksen tekeminen ja käytettävien työmenetelmien kehittäminen. Tutkimuksella etsittiin ratkaisua toimeksiantajan työajanhallintaan ja myös toteutettiin muutos. Projektin alkaessa toimeksiantaja antoi lähdeaineistoksi heidän silloin vielä KPI-seurannassaan käyttämät Excel-työkirjat sekä tiedon siitä, millaiseen muotoon kehitettävän työkalun pitää näiden sisältö muokata.

Toimeksiantajan kanssa pidettiin työkalun kehitykseen liittyviä palavereja lähes viikoittain. Tällä tavalla kertyi projektissa tarvittavaa aineistoa, eli informaatiota siitä, millaisia toiminnallisuuksista toimeksiantaja työkalulta haluaa. Projektin toimintamallista johtuen opinnäytetyössä käytettiin ei-strukturoitua haastattelumallia, koska toimeksiantajalle esitettäviä kysymyksiä ei ollut mahdollista laatia etukäteen. Syynä tähän oli se, että kysymykset olivat etupäässä palavereissa heille esitettyjä tarkentavia kysymyksiä, joiden avulla saatiin kehitystyössä tarvittavaa lisäinformaatiota. Opinnäytetyön tekijä suunnitteli, toteutti ja testasi toimeksiantajan tilaaman Excelissä toimivan KPI-työkalun vapaa-ajallaan iltaisin sekä viikonloppuisin.

2 VISUAL BASIC FOR APPLICATIONS

2.1 VBA kielen perusteet

VBA julkaistiin ensimmäisen kerran vuonna 1993 Excel 5.0 yhteydessä, mutta tällöin se ei saavuttanut kovinkaan suurta suosiota varsin rajallisten toimintojen määrän ja hankalan käytön takia. Vuonna 1997 VBA:n suosio alkoi kasvaa, sillä versiossa Excel 97 käytettävien toimintojen määrä oli moninkertaistunut ja myös koodista oli onnistuttu muokkaamaan entistäkin helppolukuisempaa (Shepherd, 2006, ix).

VBA on Visual Basic -ohjelmointikielen murre ja nämä molemmat kielet ovat Microsoftin kehittämiä. Suurin ero VBA:n ja Visual Basicin välillä on ohjelmien suorittamisessa. Visual Basic -kieltä ei voi suorittaa sellaisenaan, vaan se pitää ensin kääntää exe-tiedostomuotoon. VBA-ohjelmaa kirjoitettaessa ja suoritettaessa tarvitaan ainoastaan Excel-ohjelma, sillä versiosta Excel 97 asti VBA-koodi on sisällytetty Excel-työkirjassa olevaan VBA-editoriin (Shepherd, 2006, ix). Visual Basic -koodiin pitää ohjelmoijan osata määritellä ohjelman tarvitsemat toiminnallisuudet sekä näiden ominaisuudet huomattavasti tarkemmin kuin VBA-ohjelmaan, jossa VBA-editori osaa pitää näistä huolen ohjelmoijan puolesta (Merensalmi, 2007, 4).

VBA-koodin sisältämä työkirja on helppo tunnistaa tämän xlsm-tiedostomuodosta, joka on ollut käytössä versiosta 2007 alkaen. Tavallinen, makroja sisältämätön työkirja on ollut tästä samasta versiosta asti tiedostomuotoa xlsx.

VBA-kieli on nimensä mukaisesti visuaalista, eli pienen päättelyn ja kohtuullisen englannin kielitaidon avulla ohjelman toiminnallisuus käy ilmi tätä tutkivalle henkilölle ilman, että hän osaa vaikeaselkoista konekieltä. Tämä ominaisuus helpottaa sekä VBA-ohjelmoinnin opettelua, uusien ratkaisumallien löytämistä vaihtoehtovalikoista ja varsinkin toisten ohjelmoijien tuotoksien tarkastelua. Visuaalisuus käy ilmi myös mahdollisuutena suorittaa yksi rivi koodia kerrallaan, ja koko ajan nähdä, millaisia toimintoja ohjelma tekee käsiteltävälle aineistolle.

Alla on esimerkki VBA-koodista. Tämän toiminnallisuus kuuluu selkokielisenä seuraavasti: ensimmäisellä rivillä valitaan aktiivisen taulukon rivi 1, jonka jälkeen seuraavalla rivillä valittuna olevan alueen tekstit lihavoidaan. Esimerkki päättyy viimeisellä rivillä tapahtuvaan solun A1 aktivointiin. Tällaista toimintoa voi hyödyntää esimerkiksi jos halutaan lihavoida taulukossa olevat sarakeotsikot, jotta nämä erottuisivat paremmin varsinaisesta datasta.

```
Rows("1:1").Select
Selection.Font.Bold = True
Range("A1").Select
```

VBA-ohjelmilla voidaan toteuttaa samat toiminnot jotka käyttäjän on mahdollista tehdä itse Excelillä. Periaatteessa ohjelma suorittaa kaikki tarvittavat toiminnallisuudet aivan kuten Excelin käyttäjä tekisi nämä itse. Näiden suoritustapojen suurimmat erot käyvät ilmi sekä työnopeudessa että suorituksessa tapahtuvien virheiden määrässä. Ohjelma pystyy suorittamaan tarvittavat toiminnot kymmeniä, jopa tuhansia kertoja ihmistä nopeammin. Sen lisäksi, että VBA-ohjelma suorittaa työnsä virheettömästi, niin toiminnot tapahtuvat joka kerta täysin samalla tavalla. Näin mikään työvaihe ei jää tekemättä tai tapahdu väärässä järjestyksessä.

Vaikka Excelissä on nykyisin laaja valikoima valikoista käynnistettäviä työkaluja, kuten esimerkiksi pikasuodatus ja lajittelutoiminnot, niin silti käyttäjä ei pysty näiden avulla suorittamaan kaikkia haluamiaan toimintoja. Jos käyttäjä haluaa lisätä 20 000 riviä sisältävään dataan tyhjän rivin aina kun A-sarakkeen arvo muuttuu, tämä ei onnistu Excelin valmiilla työkaluilla. Tehokäyttäjä voi tehdä työn yhdistelemällä eri työkaluja, mutta peruskäyttäjälle tällainen säännöllisesti tehtävä työ on sekä helpointa että mukavinta suorittaa nappia painamalla – varsinkin kun työhön kuluu aikaa vain muutama sekunti.

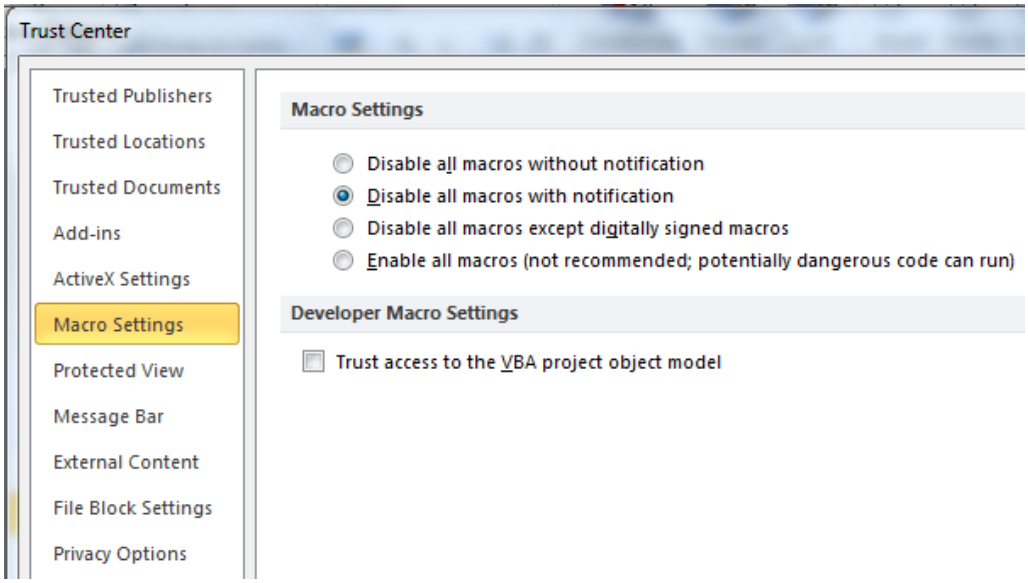
Käytännössä kukaan ei tee edellä mainitun laajuisia tehtäviä, vaan nämä jäävät tekemättä, ellei toiminnallisuutta ole automatisoitu. Tämä on huolestuttavaa, sillä työn lopputulos saattaisi olla yritykselle hyvinkin kriittinen. Nykyisin yrityksien tietojärjestelmistä pystytään ajamaan todella suuria raportteja, mutta nämä jäävät herkästi hyödyntämättä juuri näiden laajuuden vuoksi.

2.2 VBA ja tietoturva

Office-tuoteperheen ohjelmia käyttävien henkilöiden tulee ymmärtää VBA:n mahdollistamat tietoturvauhat, sillä tällä kielellä on mahdollista kirjoittaa viruksia. Vaikka käyttäjä ei itse käyttäisi VBA-ohjelmia, niin hänen koneeseensa kohdistuu vakava tietoturvan riski, mikäli hänen Excelissään on asetuksena kaikkien makrotoimintojen salliminen.

Pelkkä viruksen sisältämän työkirjan avaaminen voi riittää aktivoimaan haittaohjelman (Lehto 2009), mutta tämä voi aktivoitua pelkästään avaamalla Internet Explorer -selaimella linkki, joka osoittaa viruksen sisältämään Excel-työkirjaan (Karvonen 2003). Jos käyttäjän Excelin turvallisuusasetuksena on varoitustoiminto makroista, joka on suositeltavin vaihtoehto, niin käyttäjä voi halutessaan ottaa makrot käyttöön eikä mitään haitallista pääse automaattisesti tapahtumaan. Tästä asetuksesta ei ole kuitenkaan hyötyä, jos käyttäjä ohjelman varoituksesta huolimatta aukaisee haittaohjelman sisältävän Excel-tiedoston (office.microsoft.com).

Versiossa Excel 2010 makrojen tuvallisuusasetuksia ylläpidetään *File*-valikosta löytyvän *Options* vaihtoehdon kautta. Aukeavasta ikkunasta valitaan vaihtoehto *Trust Center*, jonka jälkeen painetaan *Trust Center Settings* -painiketta. Seuraavasta valikosta valitaan vaihtoehto *Macro Settings* ja täältä on mahdollista aktivoida suositeltu asetus *Disable all macros with notification*. Lopuksi valinta tallennetaan OK painikkeella. Tämä viimeinen avautuva ikkuna on esitelty kuviossa 1.



KUVIO 1. Makrojen turvallisuusasetuksien ylläpitäminen

Tietoturvaan liittyy ymmärrys myös siitä, että suoritetun makron toiminnallisuus ei ole peruttavissa Excelin normaalilla undo-toiminnolla, vaan makroilla suoritetut toiminnot ovat aina lopullisia. Mikäli makroa käyttämällä on muutettu joko työkirjan varsinaista dataa tai vaihtoehtoisesti muotoiluja, niin alkuperäisen tilanteen voi saada takaisin ainoastaan seuraavilla tavoilla:

- sulkemalla työkirja tallentamatta tätä, mikäli käytetty koodi ei ole sisältänyt tallennustoimintoa
- hyvällä versionhallinnalla
- ajantasaisella varmuuskopiolla

2.3 VBA:n käyttökohteita

2.3.1 Datan tarkistaminen ja etsintä

VBA-ohjelmaa kannattaa hyödyntää tarkistettaessa, että kaikki halutut arvot, esimerkiksi tuotekoodit ja näitä vastaavat hintatiedot, löytyvät myös vertailuaineistosta. Tämä toinen aineisto voi sijaita esimerkiksi aktiivisen työkirjan toisella välilehdellä tai erillisessä yrityksen verkkoasemalla sijaitsevassa tiedostossa. Vertailuaineiston sisältävän Excel-työkirjan ei tarvitse edes olla auki kun tarkastusta aletaan suorittaa, sillä VBA-ohjelmalla on mahdollista avata oikea vertailussa käytettävä työkirja. Kun tarvittava työkirja on avattu, halutut tarkastukset suoritetaan liikkuen avoimien työkirjojen välillä ja lopuksi vertailuaineistona käytetty työkirja suljetaan. Excelin peruskäyttäjä ei pysty suorittamaan edellä mainitun kaltaista tarkistusta kaavoja käyttämällä, mikäli samalle tuotenumerolle on ylläpidetty vertailuaineistoon useampia hintatietoja kampanja tms. syistä johtuen.

Rivi- ja/tai sarakeotsikkojen järjestyksen tarkistamiseen VBA sopii hyvin. Mikäli jokin raportti ajetaan säännöllisesti yrityksen tietojärjestelmästä ja tämä raportti liitetään osaksi jo olemassa olevaa raporttia, on tällöin huolehdittava raporttien yhdenmukaisuudesta. Mikäli järjestelmästä otetun raportin otsikkotiedot eroavat kohderaportin otsikoista, on mahdollista, että kohderaporttiin ylläpidettävät tiedot kohdistuvat väärille otsikoille.

2.3.2 Datan muokkaaminen

Yrityksen tietojärjestelmä voi olla sellainen, että raportille tulostuu ei-toivottuja rivi- ja/tai sarakeotsikoita, joiden rivit ja/tai sarakkeet halutaan poistaa aineistosta. Toiminto voidaan suorittaa automaattisesti siten, että ohjelman koodiin kirjoitetaan poistettavat otsikkonimet. Näiden ylläpidettyjen nimien avulla voidaan datasta poistaa kaikki rivit ja/tai sarakkeet, joiden otsikkonimi vastaa koodissa olevaa tietoa. Vaihtoehtoisesti poisto voi tapahtua myös siten, että rivi ja/tai sarake poistetaan, mikäli tämän otsikkoa ei ole määritetty säästettäväksi.

Poiston voi tehdä myös määrittäen poistettavien rivien ja/tai sarakkeiden numerot. Huomioitavaa on, että ihmisille saraketunnisteet ovat kirjaimia, mutta VBA käsittelee nämä numeroina. Sarake A on 1, sarake B on 2 jne. Tämä toimintamalli on kuitenkin virheherkkä, mikäli on mahdollista, että järjestelmästä tulostettavan raportin sisältö vaihtelee. Tällöin ohjelma saattaa poistaa raportilta tärkeän rivin tai sarakkeen ja jättää aineistoon poistettavaksi tarkoitetut tiedot. Jos raportti on määrämuotoinen, niin tarvittava koodi on mahdollista toteuttaa, vaikka käyttäjä ei osaisi kirjoittaa VBA-koodia. Tällöin koodi toteutetaan VBA:n nauhoitustoiminnolla, jonka toiminnallisuus esitellään kappaleessa 2.4.1. Myös ne rivit ja/tai sarakkeet voidaan poistaa, jotka eivät täytä annettuja ehtoja. Datasta voidaan esimerkiksi poistaa kaikki rivit, joiden arvo on A-sarakkeessa pienempi kuin 100.

Ohjelmalla voidaan vastaavasti lisätä rivejä ja/tai sarakkeita haluttuihin kohtiin taulukkoa, sekä nimetä näiden lisättyjen rivien ja/tai sarakkeiden otsikot annettujen määritysten mukaisiksi. Raportille tulostuneet otsikot voidaan tarvittaessa vaihtaa makrotoiminnon avulla informatiivisemmiksi kuin mitä yrityksen tietojärjestelmästä on saatavissa.

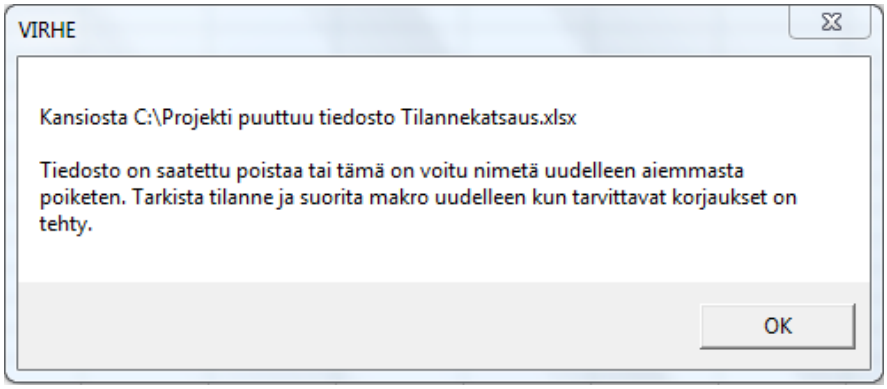
Kaavojen lopputulokset voidaan muuttaa teksti-, luku- tms. muotoon. Tämä on tärkeää jos halutaan, että kaavojen lopputulokset eivät muutu datan muuttuessa. Kaava saattaa joissain tapauksissa viitata aktiivisen työkirjan ulkopuolelle, johon työkirjan loppukäyttäjän oikeuksilla ei ole pääsyä. Tällaisessa tapauksessa kaavan lopputulos on virheilmoitus halutun lukuarvon tai tekstin sijasta. Virhetilanne tapahtuu varmuudella, mikäli yrityksen ulkopuolelle lähetetään sähköpostitse viittaavia kaavoja sisältävä työkirja.

2.3.3 Laskukaavojen lisääminen

VBA:lla on mahdollista syöttää taulukoihin samoja kaavoja kuin käsin kirjoittamalla. Tämä on hyödyllistä varsinkin kun syötettävä kaava on pitkä tai työkirjan ylläpitäjälle kaavojen kirjoittaminen on liian haasteellista. Kaavojen lisääminen VBA:lla on kannattavaa esimerkiksi silloin, kun yrityksen tietojärjestelmästä ajetaan säännöllisesti summarivin sisältävä raportti, josta poistetaan summan laskennassa käytettyjä rivejä. Mikäli uutta summariviä ei lasketa raportille, jäävät tähän alkuperäiset summat, joiden lopputulokset ovat muuttuneet virheellisiksi rivien poiston myötä. VBA:lla voidaan hallinnoida haluttujen rivien poisto, uuden tarvittavan summarivin teko laskukaavalla ja lopuksi ohjelma voi muuttaa kaavojen tulokset luvuiksi kaavojen sijasta.

2.3.4 Virhetilanteiden hallinta

Oikeaoppisesti kirjoitettujen ohjelmien tulee antaa käyttäjälle kuvaava virheilmoitus kaikissa mahdollisissa virhetilanteissa, ja pysäyttää ohjelman suorittaminen hallitusti. Näin käyttäjät ymmärtävät millaisesta virheestä on kysymys ja osaavat tarvittaessa tehdä tiedostoihinsa ja/tai ohjelmaan tilanteen vaatimat muutokset. Virhetilanne voi olla esimerkiksi tarvittavan lähdedatan puuttuminen aineistosta tai muu tilanne, jota ohjelma ei pysty suorittamaan. Mikäli ohjelma yrittää avata määritettyä Excel-työkirjaa, jota ei löydy kohdekansiosta, on kyseessä virhetilanne. Tällöin käyttäjälle on pystyttävä antamaan selkeä ilmoitus, jotta hän osaa tehdä tarvittavat tarkistus- ja korjaustoiminnot. Kuviossa 2 annetaan käyttäjälle tieto tapahtuneesta virheestä ja neuvotaan miten hänen tulee toimia kyseisessä tilanteessa.



KUVIO 2. Kuvaavan virheilmoituksen antaminen käyttäjälle

2.4 VBA-koodin tuottaminen

2.4.1 Koodin nauhoittaminen

Koodin nauhoittamisella tarkoitetaan VBA-koodin luomista siten, että käyttäjä painaa Excelissä olevaa *Record Macro* -painiketta ja tämän jälkeen VBA-editori muuntaa käyttäjän tekemät toiminnallisuudet VBA-koodiksi. Nauhoituksen käynnistyessä Excel muodostaa automaattisesti tarvittavan makromoduulin VBA-editoriin ilman, että käyttäjän tarvitsee suorittaa tarvittavia toimintoja itse. Nauhoittamalla syntyvä koodi tallentuu automaattisesti tähän luotuun moduuliin. Nauhoittaminen päättyy vasta kun käyttäjä itse pysäyttää nauhoituksen.

Nauhoitustoimintoa käytettäessä VBA-koodiksi muuntuvat ainoastaan käyttäjän tekemät merkitsevät toiminnot. Tällaisia ovat kaikki toiminnot, joiden pohjalta ohjelmassa tapahtuu jonkinlainen muutos. Mikäli käyttäjä klikkaa useita soluja peräjälkeen aktiiviseksi tekemättä kuitenkaan muita toimintoja, ainoastaan viimeisen solun aktivointi ennen todellisen merkittävän toiminnon suorittamista tallentuu koodiin. Merkittäviä toimintoja ovat esimerkiksi solun arvon muuttaminen tai rivin lisääminen työkirjaan.

Nauhoittamalla voi tehdä koodia ainoastaan sellaisia tehtäviä varten, jotka suoritetaan aina täysin identtisellä tavalla. Tällainen tehtävä on esimerkiksi lomakkeen tyhjentäminen tämän syöttösoluihin täytetyistä tiedoista. Käyttäjän nauhoittamiseen käyttämällä ajalla ei ole vaikutusta syntyvään VBA-koodiin, ainoastaan hänen tekemänsä toiminnot ovat merkitseviä. Vaikka käyttäjä nauhoittaisi vain muutamia toiminnallisuuksia sisältävää kokonaisuutta koko yön ajan, niin ohjelmaa suorittaminen kestää Exceliltä ainoastaan muutamia sekunnin kymmenyksiä.

Nauhoittamalla toimintoja ohjelmoija voittaa aikaa ja voi varmistua koodinsa oikeellisuudesta. Koodi, joka on ohjelman itsensä luomaa, toimii 100 % varmuudella teknisestä näkökulmasta tarkasteltuna. Ohjelmoijan vastuulle jää, että hän on nauhoittanut koodin haluttujen toimintojen vaatimalla tavalla.

2.4.2 Koodin kirjoittaminen

Myös kirjoittamalla tuotetut VBA-ohjelmat sekä kirjoitetaan että tallennetaan VBA-editorissa oleviin makromoduuleihin. Mikäli koodia kirjoitetaan sellaiseen työkirjaan, jossa ei ole entuudestaan makroja, kannattaa ohjelmoijan tällöin käynnistää makron nauhoitustoiminto, mutta lopettaa se välittömästi. Näin toimimalla välttyy tarvittavan makromoduulin luomiselta, kun VBA-editori luo sen ohjelmoijan puolesta.

Excelin nauhoitustoiminto on niin helppokäyttöinen, ettei ohjelmoijan kannata kirjoittaa itse VBA-koodia ellei ole pakko. Kirjoittamalla on tehokasta tehdä ainoastaan lyhyet komennot, joita ei ennestään esiinny koodissa ja sellaiset toiminnot, joita ei voi toteuttaa nauhoitustoiminnolla. Nauhoittamalla ei voi tuottaa esimerkiksi silmukkatoimintoja tai ehtolauseita.

Koodia kirjoittaessakin on siis mahdollista hyödyntää nauhoitustoimintoa. Ohjelmoija voi toteuttaa nauhoittamalla kaikki ohjelmassa tarvittavat perustoiminnallisuudet ja kopioida näin syntyvä koodi kehitettävään ohjelmaan. Nauhoitetusta koodista on helppo poistaa turhat toiminnallisuudet ja lisätä sellaisia, joita ei ole nauhoittamalla mahdollista toteuttaa.

2.4.3 Ehtolauseet

Koodia kirjoittaessa tulee väistämättä eteen tilanteita, että ohjelmalle on pystyttävä antamaan vaihtoehtoja. Yksinkertaisimmillaan ehtolause on alla olevan esimerkin IF-lause, jonka ensimmäisellä rivillä ehto kuuluu seuraavasti: ”Jos solun A1 arvo on tyhjä, niin…”. Saapuessaan tälle riville ohjelma tarkistaa solun A1 arvon, ja mikäli arvo on tyhjä, niin tällöin ohjelma menee IF-lauseen sisälle. Esimerkin tapauksessa IF-lauseen sisällä oleva komento lisää merkin *X* aiemmin tyhjänä olleeseen soluun A1. Mutta mikäli solun A1 arvo on mikä tahansa muu kuin tyhjä, ohjelma siirtyy suoraan ensimmäiseltä riviltä *End If* -komennon jälkeiselle riville.

```
If Range("A1").Value = "" Then
    Range("A1").Value = "X"
End If
```

Mikäli mahdollisia vaihtoehtoja on useampia, ohjelmoijan on yksinkertaisinta käyttää monivalintalausetta, sillä tällä menetelmällä tarvittavan koodin määrä on vähäisempää kuin IF-lauseella toteutettuna. Myös uusien vaihtoehtojen lisääminen monivalintalauseeseen on jälkeenpäin helpompaa kuin IF-lauseen muokkaaminen.

Vapaavalintaisesti nimettävä muuttuja, jonka nimi on seuraavassa esimerkissä *vaihtoehdot*, saa ensimmäisellä rivillä arvokseen solun A1 arvon. Tämän jälkeen alkaa varsinainen monivalintalause *Select Case*, jonka perässä mainitaan aina kohteen nimi eli muuttuja. Monivalintalauseella voidaan helposti hallinnoida isoakin vaihtoehtojen kokonaisuutta, esimerkiksi virhenumeroiden perusteella käyttäjälle annettavia virheilmoituksia. Monivalintalause päättyy komentoon *End Select*.

```
vaihtoehdot = Range("A1").Value

Select Case vaihtoehdot

    Case "Kyllä"
        Range("A1").Value = "Yes"

    Case "Ei"
        Range("A1").Value = "No"

    Case Else
        Range("A1").Value = ""

End Select
```

Mikäli edellisessä esimerkissä muuttujan *vaihtoehdot* arvo on *Kyllä*, monivalintalause muuttaa solun A1 arvoksi sanan *Yes*. Jos muuttujan arvo on *Ei*, solun A1 arvoksi tulee *No*. Muuttujan arvon ollessa mikä muu tahansa, solu A1 tyhjennetään.

2.4.4 Silmukat

Silmukat mahdollistavat saman ohjelman hyödyntämisen lähdeaineistoille, joiden koko vaihtelee. Silmukoita käyttäessä ei ole merkitystä, onko aineistossa vain muutamia vai satoja tuhansia rivejä dataa. Ainoan rajoituksen tuo tällöin käytettävä fyysinen laitteisto.

FOR-silmukkaa suoritetaan kunnes silmukan sisällä oleva muuttuja on yhtä suuri tai suurempi kuin silmukan aloitusrivillä määritetty loppuarvo. Jokainen silmukan pyörähdys kasvattaa muuttujan arvoa yhdellä. Kuvion 3 ohjelma tuo käyttäjälle jokaisella pyörähdyksellä sanomaikkunan, jossa näkyvä arvo kasvaa jokaisella pyörähdyksellä yhdel-

lä. Ensimmäisellä kerralla muuttujan arvo on yksi, toisella kerralla kaksi jne. kunnes arvo on kymmenen. Tämän jälkeen ohjelma poistuu silmukasta jatkaen ohjelman suorittamista. Sanomaikkunoiden toiminnallisuus esitellään kappaleessa 2.4.5.

```
For arvo = 1 To 10
    MsgBox (arvo)
Next arvo
```

KUVIO 3. FOR-silmukka

Do-silmukoita on erilaisia, mutta yhteistä näille on se, että nämä alkavat komennolla *Do* ja päättyvät komentoon *Loop*. Aloittava komento voi olla pelkkä *Do*, mutta tällöin silmukan sisällä on oltava ehto, jonka täytyttyä ohjelma osaa poistua silmukasta. Mikäli mainittua ehtoa ei määritetä, ohjelma jää pyörimään ikuiseen silmukkaan, joka johtaa lopulta joko virhetilanteeseen tai mahdollisesti koko Excelin kaatumiseen. Excel tarkastaa suoritettavan koodin validiteetin ennen VBA-ohjelman suorittamista, mutta ikuisia silmukoita tämä tarkastus ei havaitse.

Do until -komento on ehkä VBA:ssa yleisimmin hyödynnetty silmukka. *Do until* -komennon perään määritetään ehto, jonka täytyttyä silmukan suorittaminen päättyy. Seuraavassa esimerkissä ohjelma suorittaa silmukkaa kunnes komento *Loop* palauttaa ohjelman suorittamisen silmukan ensimmäiselle riville hetkellä, jolloin aktiivisen solun arvo on tyhjä. Tällöin silmukan suorittaminen päättyy välittömästi ja ohjelma hyppää komennon *Loop* jälkeiselle riville jatkaen tästä ohjelman suorittamista.

```
Do Until ActiveCell.Value = ""
    ActiveCell.Offset(1, 0).Select
Loop
```

2.4.5 Käyttäjän kanssa kommunikointi

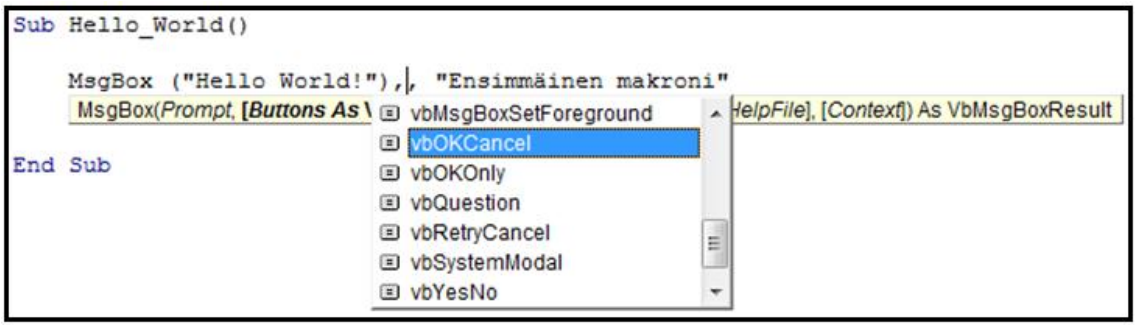
Sanomaikkunoilla on mahdollista informoida käyttäjää, esimerkiksi virheilmoituksia näyttämällä. Kuvion 4 ohjelmalla voidaan toteuttaa *Hello World!* -ohjelma, joka avaa käyttäjälle Kuvion 5 mukaisen sanomaikkunan. Tässä sanomaikkunassa on tekstin lisäksi ainoastaan OK-painike, jota painamalla sanomaikkuna sulkeutuu.

```
Sub Hello_World()

    MsgBox ("Hello World!")

End Sub
```

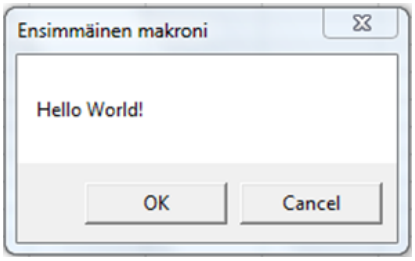
KUVIO 4. Hello World! –ohjelman koodi

KUVIO 5. Hello World! -ohjelman muodostama sanomaikkuna

Sanomaikkunaan voidaan tarvittaessa liittää otsikko sekä lisätä painikkeita, mikäli käyttäjälle halutaan antaa muitakin vaihtoehtoja kuin pelkkä *OK*. Usein on tarvetta esimerkiksi *Cancel*-painikkeelle, jotta käyttäjä voi estää jonkin toiminnallisuuden tapahtumisen. Seuraavassa esimerkissä on tilanne, jossa edellisen esimerkin koodia ollaan laajentamassa VBA:n avustustoimintoa käyttäen.



Yllä oleva koodi on muuten täysin sama kuin edellisessä esimerkissä, mutta tähän on lisätty otsikko *Ensimmäinen makro* ja tähän ollaan juuri tekemässä valintaa OK ja Cancel -painikkeille. Huomioitavaa on, että painikkeille on koodattava haluttu toiminnallisuus, muuten ainoa tapahtuma on sanomaikkunan sulkeutuminen. Koodin alla näkyy syöttöä helpottamassa keltainen infopalkki, joka ilmaisee mitä kaikkia tietoja kyseiseen sanomaikkunaan on mahdollista ylläpitää. Lisäksi lihavoituna näkyy tieto, jota ollaan juuri syöttämässä. Esimerkin tapauksessa lihavoituna on teksti *Buttons As*, jolloin ohjelmoija osaa päätellä olevansa määrittelemässä sanomaikkunaan tulevia painikkeita. Edellä mainituilla lisäyksillä sanomaikkuna näyttää kuvion 6 mukaiselta.

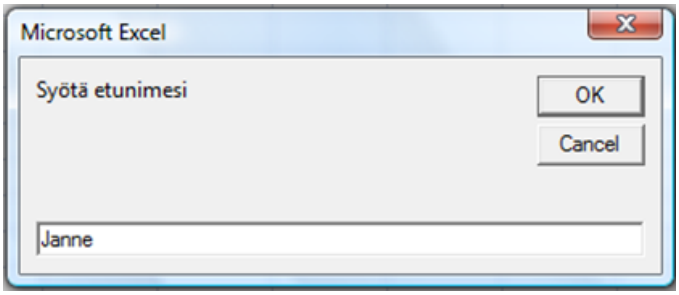


KUVIO 6. Sanomaikkuna otsikolla ja lisäpainikkeella

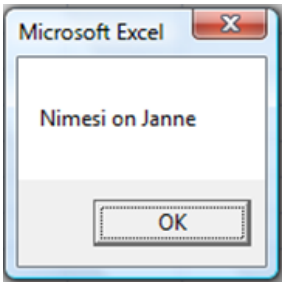
Syöttöikkunan yhteydessä tulee käyttää muuttujaa, jonka arvo määräytyy käyttäjän antaman syötteen perusteella. Seuraavassa esimerkissä muuttuja *Nimi* saa käyttäjän syöttämän arvon ja seuraavalla rivillä tätä muuttujaa hyödynnetään osana käyttäjälle näytettävän sanomaikkunan tekstiä.

```
Nimi = InputBox("Syötä etunimesi")

MsgBox ("Nimesi on " & Nimi)
```

Kuviossa 7 on kuva syöttöikkunasta, käyttäjän syötettyä siihen nimensä. Kuviossa 8 esitellään sanomaikkuna, joka on toteutettu yllä olevan esimerkin kahdella koodirivillä.



KUVIO 7. Syöttöikkuna, johon käyttäjä on lisännyt syötteen



KUVIO 8. Sanomaikkuna, jossa on hyödynnetty käyttäjän antamaa syötettä

2.4.6 Viittaaminen työkirjan sisäiseen välilehteen

Koodin suorittamisen aikana täytyy usein vaihtaa välilehteä, jotta ohjelma voi hakea täältä tarvitsemaansa tietoa. Riippuen suoritettavasta toiminnallisuudesta sekä ohjelmoijan omista tottumuksista, ohjelmaa suoritettaessa voidaan liikkua aktiivisen tiedoston sisäisillä välilehdillä kolmella vaihtoehtoisella tavalla. Nämä on esitelty kuviossa 9.

```
'1
Sheets("Etusivu").Select

'2
x = ActiveSheet.Name
x = "Etusivu"
Sheets(x).Select

'3
ActiveSheet.Next.Select
ActiveSheet.Previous.Select
```

KUVIO 9. Vaihtoehtoiset tavat siirtyä välilehdeltä toiselle

1) Siirrytään välilehdelle tämän nimen perusteella
   Tämä on yksinkertaisin esimerkin komennoista ja siirtyminen tapahtuu välittömästi, vaikka työkirjassa olisi hyvinkin paljon välilehtiä.

2) Välilehti valitaan muuttujan avulla
   Ohjelmalle tulee kertoa muuttujan arvo ennen tämän hyödyntämistä. Muuttujan arvo voidaan kertoa ohjelmalle aktiivisen välilehden perusteella koodia aiemmin suoritettaessa, kuten ensimmäisellä rivillä on tehty. Vaihtoehtoisesti muuttujan arvon voi kirjoittaa suoraan koodiin keskimmäisen esimerkin mukaisesti. Alimmalla rivillä on varsinainen siirtokomento, joka on lähes identtinen osioon 1 verrattuna, mutta muuttujan ollessa kyseessä ei käytetä lainausmerkkejä.

3) Aktiivisen välilehden viereiselle sivulle siirtyminen
   Ensimmäisessä esimerkissä siirto tapahtuu välilehdelle, joka sijaitsee aktiivisen välilehden oikealla puolella. Tätä komentoa käytettäessä ei tarvitse edes tietää kyseisen välilehden nimeä. Vastaavasti vasemmalla puolella olevalle välilehdelle siirrytään jälkimmäisellä vaihtoehdolla, esimerkiksi palatessa takaisin alkuperäiselle sivulle, kun ensin on käytetty ensimmäisen rivin komentoa.

Kaikki edellä mainitut komennot ovat mahdollisia ainoastaan näkyville välilehdille siirryttäessä. Mikäli työkirjalla on piilotettuja välilehtiä, joille ohjelman tulee siirtyä, nämä välilehdet on muutettava näkyviksi siirtymisen mahdollistamiseksi.

2.4.7 Ulkoisten työkirjojen hallinta

VBA:lla on mahdollista liikkua myös aktiivisen työkirjan ulkopuolella. Usein tarvittava data ei ole saatavilla aktiivisella työkirjalla, jolla varsinainen tarkastus tai muu toiminto suoritetaan, vaan tiedot haetaan ulkoiselta, mahdollisesti suljetulta, työkirjalta. Mikäli haku tapahtuu ohjelman suorittamana suljetusta Excel-tiedostosta, on tämä mahdollista ainoastaan, mikäli ohjelman käyttäjällä on pääsyoikeudet kyseiseen tiedostoon. Vaikka siirtyminen ulkoiselle työkirjalle tapahtuu automaattisesti, ja ehkä jopa käyttäjän huomaamatta, poikkeaa toiminto vain nopeudeltaan käyttäjän itse suorittamasta työstä.

VBA-ohjelma joutuu ihmisen tavoin avaamaan halutun Excel-tiedoston ennen kuin se pääsee työkirjan sisältöön käsiksi. Ulkoiselle työkirjalle voidaan myös tallentaa tietoja VBA-ohjelman avulla. Tällöin koodissa tulee olla ennen kohdetiedoston sulkemista tallennuskomento, jotta tiedot tallentuvat tiedostoon ja tiedoston sulkeminen onnistuu. Ulkoinen työkirja avataan komennolla *Workbooks.Open*, mutta ennen tiedoston avaamista ohjelman tulee tietää missä kansiossa se sijaitsee. Tämän tiedon ohjelma saa komennon *ChDir* avulla. Avattavan tiedoston nimen ja/tai kohteen tallennuskansion tilalla voi hyödyntää muuttujaa aiempien esimerkkien tavoin.

```
ChDir "C:\Projekti"
Workbooks.Open ("Aikataulu.xlsx")
```

Jotta ulkoisen työkirjan avaamisesta saadaan täysi hyöty irti, tämän ja alkuperäisen työkirjan välillä pitää pystyä liikkumaan. Tämä tapahtuu periaatteessa aivan samoin kuin aktiivisen työkirjan sisällä eri välilehdillä liikuttaessa. Seuraavan esimerkin komennolla ohjelma siirtyy työkirjalle, jonka nimi on *Hinnasto*. Myös työkirjojen nimien hallinnointi onnistuu muuttujien avulla.

```
Workbooks("Hinnasto").Activate
```

2.4.8 Aliohjelmat

VBA-ohjelmoinnissa voidaan hyödyntää olio-ohjelmoinnista tuttuja aliohjelmia. Näin vähennetään tarvittavan koodin määrää, kun ohjelmassa toistuvat toiminnot voidaan suorittaa yhdellä ohjelmalla sen sijaan, että tarvittavat koodit toistuisivat ohjelmassa yhtä usein kuin näitä tarvitaan. Aliohjelmat jaetaan kahteen ryhmään näiden toiminnallisuuden perusteella. Mikäli aliohjelma palauttaa isäntäohjelmaan suorittamansa toiminnallisuuden perusteella jonkin lopputuloksen, kyseessä on *funktio*. Kun aliohjelma suorittaa ainoastaan tietyn toiminnallisuuden palaten tämän jälkeen isäntäohjelmaan, niin tähän riittää *proseduuri*, jollaisesta on kuviossa 10 yksinkertaisin mahdollinen kuvaus.

```
Sub proseduuri()

    '1
    aliohjelma

    '3

End Sub

Public Function aliohjelma()

    '2
    Range("A1").Select

End Function
```

KUVIO 10. Proseduuri-tyyppinen aliohjelma

Pääohjelmaa suoritetaan kunnes tämä saapuu aliohjelmaa kutsuvalle riville (1). Esimerkissä käytetylle aliohjelmalle on annettu yksinkertainen ja kuvaava nimi *aliohjelma*, josta koodi hyppää suorittamaan samannimistä proseduuria (2). Proseduurin suorittaminen kestää kunnes kaikki tämän rivit on käyty läpi. Näitä voi olla proseduurin tarkoituksesta riippuen kymmeniä tai jopa satoja. Kun proseduuri on suoritettu loppuun, ohjelma palaa pääohjelmaan siihen kohtaan, josta proseduuria kutsuttiin (3), jatkaen tästä ohjelman suorittamista.

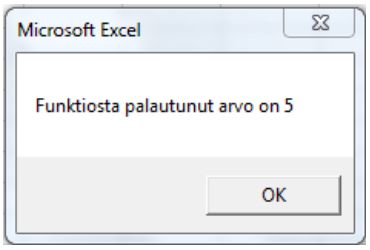
Pääohjelma ei siis tiedä proseduurin toiminnasta muuta, kuin että sen suorittaminen on joko kesken tai jo päättynyt, jolloin pääohjelman suorittaminen voi jatkua. Koska proseduurit eivät pysty palauttamaan tietoa pääohjelmaan, voidaan näitä hyödyntää toiminnallisiin tehtäviin, jollaisia on esimerkiksi taulukon rivien tai sarakkeiden poistaminen ja lisääminen (Merensalmi, 2007, 58).

VBA-koodissa ei yleensä tarvitse esitellä muuttujia, mutta apuohjelmia käytettäessä tämä on välttämätöntä, mikäli aliohjelmalle annetaan muuttujan avulla informaatiota. Kuvioissa 11 on esitelty funktion toiminnallisuus, joka on periaatteessa kuten proseduuri, mutta funktio palauttaa haluttua tietoa pääohjelmaan käytettävän muuttujan avulla (Merensalmi, 2007, 58).

```
Sub funktio()

    '1
    Dim x, y As Integer

    '2
    x = 2
    y = 3

    '3 & 5
    i = aliohjelma(x, y)

    '6
    MsgBox ("Funktiosta palautunut arvo on: " & i)

End Sub

Public Function aliohjelma(x, y As Integer)

    '4
    aliohjelma = x + y

End Function
```

KUVIO 11. Funktio-tyyppinen aliohjelma

Edellisessä esimerkissä funktio laskee yhteen pääohjelmasta ilmoitettavia lukuja. Tieto- ja kuljettavat muuttujat esitellään (1) ennen kuin näille annetaan arvot (2). Seuraavaksi kutsutaan funktiota nimeltä *aliohjelma*, jonka mukana esitellyt muuttujat toimitetaan funktioon (3). Funktiota suoritetaan kuten proseduuria, eli kunnes aliohjelma päättyy. Oleellista on, että ohjelman lopuksi muuttuja, joka on samanniminen kuin itse *funktio*, saa pääohjelmaan toimitettavan arvon (4). Tässä esimerkissä arvo on muuttujien x ja y summa. Tämän jälkeen funktiota kutsuneella rivillä oleva muuttuja saa arvon, joka palautuu funktiosta (5). Lopuksi palautettua arvoa voidaan hyödyntää pääohjelmassa, ja esimerkissä muuttujien x ja y summa näytetään sanomaikkunan tekstinä (6). Tämä on esitelty kuviossa 12.



KUVIO 12. Sanomaikkuna, jossa näytetään funktiolla laskettu arvo

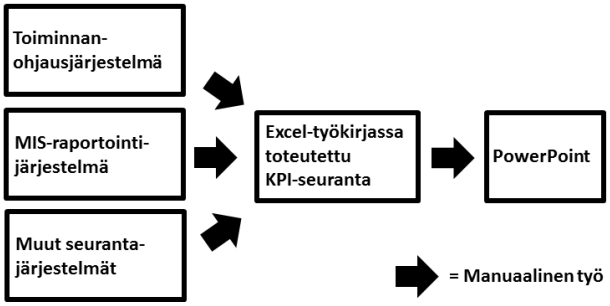
3 CASE KPI-TYÖKALU

3.1 Lähtötilanne

Projektin lähtötilanne oli haasteellinen, sillä toimeksiantaja oli vasta aloittamassa KPI-mittaristonsa uusimisen. Tämä tarkoitti käytännössä sitä, että vielä projektin alkaessa suuri osa heidän käyttämistä lähdeaineistoistaan tulisi projektin edetessä jäämään pois käytöstä, ja että näiden tilalle tulisi vastaavasti paljon uusia aineistoja. Ohjelman kehittäjän kannalta tämä ei ollut mukava tilanne, sillä projektin alkaessa oli aloitettava työkalun kehitystyö tietoisena siitä, että osa tehdystä työstä tulee väistämättä olemaan turhaa. Tärkeintä oli kuitenkin saada jo ennen työhön ryhtymistä tieto siitä, että työ tapahtuu jatkuvasti muuttuvassa ympäristössä.

Projekti käynnistettiin aloituspalaverilla, jossa toimeksiantaja toimitti kaikki heidän vielä tuolloin KPI-seurannassa käyttämänsä lähdeaineistot sekä tiedon siitä, mihin muotoon mikäkin näistä aineistoista tulee muokata. Samalla sovittiin myös, että projektiin liittyviä palavereja pyritään pitämään viikoittain tai tarvittaessa useamminkin, sillä uusien aineistojen läpikäynti on helpompaa kasvokkain kuin sähköpostitse. Näin toimittaessa oli helpompi antaa toimeksiantajalle tilannekatsaus kehitettävästä työkalusta.

Toimeksiantajan KPI-seurantaprosessi oli aiemmin toteutettu täysin Excelillä ja PowerPointilla suoritettavana käsityönä. Tämä prosessi esitellään kuviossa 13. Seurannassa käytetyt lähdeaineistot ajettiin yksi kerrallaan eri järjestelmistä ja siirrettiin Excel-työkirjaan, jossa sisällön tarkastuksen jälkeen aineistoista piirrettiin graafiset kuvaajat kunkin mittarin vaatimusten mukaisesti. Lopuksi kuvat siirrettiin PowerPoint-tiedostoon, jota käytettiin KPI-lukujen varsinaisena tiedotus- ja seurantatyökaluna.



KUVIO 13. Kuvaus toimeksiantajan aiemmasta KPI-seurantaprosessista

Toimeksiantajan tuleva KPI-raportointijärjestelmä on projektin päättymisen jälkeen ulkopuolisen toimittajan toteuttama Advanced Excel -järjestelmä, jonka käytöstä edes toimeksiantajalla ei ollut projektin alkaessa käyttökokemusta. Koska kehitettävä KPI-työkalu tulee mahdollistamaan tulevan raportointijärjestelmän tehokkaan käytön ja näiden ohjelmien prosessit ovat voimakkaasti yhteydessä toisensa, tuli KPI-työkalun kehittäjän toimia projektin ajan myös tämän KPI-raportointijärjestelmän ylläpitäjänä.

3.2 Projektin tavoitteet

Toimeksiantaja halusi tehostaa kaikki KPI-mittaamiseen liittyvät prosessit samalla kun heidän koko KPI-mittaristonsa uusittiin. Aiemmin koko KPI-seuranta on tehty Excelillä ja PowerPointilla käsin ja työ on vienyt kuukausittain yhteensä 1,5 - 2,5 työpäivää yhdeltä johtoryhmän jäseneltä sekä yhdeltä Business Controllerilta. Projektin tavoitteena oli kehittää toimeksiantajalle VBA-ohjelma, joka automaattisesti sekä tarkistaa että muokkaa määritettyyn muotoon kaikki KPI-seurannassa käytettävät Excel-työkirjat ennen näiden sisältämän datan liittämistä varsinaiseen KPI-raportointijärjestelmään. Projektin tärkein tavoite oli vapauttaa työn aiemmin suorittaneet henkilöt todellista asiantuntemusta vaativien tehtävien pariin, kuten esimerkiksi KPI-lukujen analysointiin sekä näiden perusteella tehtävien päätösten tekoon.

Tilatun työkalun tuli olla niin helppokäyttöinen, että Excelin peruskäyttäjä pystyy käyttämään sitä pelkkien kirjallisten käyttöohjeiden avulla ilman VBA-ohjelmoinnin tuntemusta tai muita erityistaitoja. Vaatimuksissa oli myös määritelty, että työkalun tulee olla toimintavarma, jotta minkäänlaisia tästä johtuvia raportointivirheitä ei pääse esiintymään. Työkalulta vaadittiin myös skaalautuvuutta, eli tulevaisuudessa KPI-seurantaan lisättävien uusien lähdeaineistojen käsittelyn pitää onnistua ilman ohjelmaan tehtäviä muutoksia. Myös työkalun ylläpitäjän pitää pystyä lisäämään tähän tulevaisuudessa tarvittavat Excel-työkirjat, tukenaan pelkästään kirjalliset ohjeet.

Opinnäytetyön tekijän henkilökohtaisena tavoitteena oli kehittää KPI-työkaluun ainoastaan yksi VBA-ohjelma, jota hyödynnetään kaikille seurannassa käytettäville lähdeaineistoille huolimatta näiden muotoiluista. Tämän tavoitteen toteutuessa työkalun käyttäjäystävällisyys kasvaisi huomattavasti. Tällöin ohjelman ylläpitäjän ei tarvitsisi tietää,

mikä ohjelma hänen tulee kohdistaa uudelle työkaluun lisättävälle lähdeaineistolle. Lisäksi tekijän tavoitteena oli mitata ohjelman kehittämiseen kuluva aika, jotta osana tätä raporttia on mahdollista kuvata, paljonko näin laajamittaiseen työhön kuluu aikaa.

3.3 Suunnittelu

Työkalun toiminnallisuuden suunnittelu lähti aikeesta kehittää ainoastaan yksi ohjelma, joka käsittelee kaikki tarvittavat lähdeaineistot, kuten opinnäytetyön tavoitteeksi oli asetettu. Tulevaa ohjelmaa ei kuitenkaan ollut mahdollista suunnitella täysin ennen kehitystyön aloittamista, sillä projektin alkaessa ei ollut tietoa jatkossa ohjelman käsiteltäväksi lisättävien Excel-työkirjojen muotoilusta. Syynä tähän oli toimeksiantajan MIS-järjestelmä, josta suurin osa KPI-seurannassa tarvittavasta lähdedatasta otetaan, mutta jonka raporttien muotoiluun ei ole mahdollista täysin vaikuttaa. Raportin sisällöstä riippuen tämän muotoilu saattaa olla hyvinkin poikkeava muihin raportteihin verrattuna. Ilman tietoa tulevien raporttien muotoiluista, ei ollut mahdollista kehittää ohjelmaan näiden muokkaamiseksi vaadittavia toiminnallisuuksia. Projektin alkaessa ei ollut tietoutta myöskään siitä, kuinka paljon lähdedataa sisältäviä Excel-tiedostoja toimeksiantajan KPI-mittaristo tulee tarvitsemaan toimiakseen.

Suunnittelun pohjana oli myös visio, että kaikki seurannassa tarvittavat tiedostot tultaisiin tallentamaan yhteen ennalta sovittuun kohdekansioon, josta ohjelma löytäisi nämä. Tästä ajatuksesta kuitenkin luovuttiin toimeksiantajan edustajan ehdottaessa useamman tallennuskansion käyttöönottoa. Syynä ehdotukseen oli halu helpottaa KPI-seurannassa käytettävien lähdeaineistojen tallentamista, sillä tämä työ toteutetaan toimeksiantajan henkilökunnan toimesta. Ehdotus oli perusteltua myös kehitettävän ohjelman toiminnallisuuden turvaamiseksi. Mikäli väärä aineisto tallennettaisiin kansioon ohjelman tunnistamalla nimellä, niin tällöin tapahtuisi joko virhetilanne työkirjan muotoilujen ollessa vääriä, tai pahimmassa tapauksessa työkalu saattaisi pystyä käsittelemään työkirjan tiedot. Tällöin virheen havaitseminen jäisi joko tietoja KPI-seurantajärjestelmän palvelimelle syötävän käyttäjän tai KPI-lukuja analysoivan henkilön varaan. Mutta nykyisin yhden tallennuskansion sisältäessä keskimäärin vain 8 tiedostoa, sekaannuksen määrä tallennusta tehtäessä on hyvin pieni.

Edellä mainittujen seikkojen takia suunnittelua toteutettiin jatkuvasti projektin edetessä. Ohjelman toiminnallisuutta suunniteltiin aina toimeksiantajan toimittaessa uusia lähdeaineistoja, joiden muotoilu poikkesi aineistoista, joiden muokkaustoiminnot oli jo ylläpidetty ohjelmaan. Tarve uusille toiminnallisuuksille aiheutti poikkeuksetta muutoksia myös jo kehitettyyn ohjelmaan. Muutosten toteuttamisen lisäksi aikaa kului tehtyjen muutosten testaamiseen kaikilla työkalussa käytettävillä erimuotoisilla lähdeaineistoilla.

Ensimmäisten työkalun kehitystä käsittelevien palaverien aikana havaittiin, että tuleva työkalu tarvitsee nimen, jotta tästä on helpompi keskustella. Koska työkalun tehtävänä on tarkistaa, että KPI-raportointijärjestelmään pyrkivä data on siinä kunnossa, että se voidaan päästää sisään, niin tämän työnkuvauksen todettiin vastaavan läheisesti ravintolan ovimiesten työtehtäviä, heidän tarkastaessaan omien asiakkaidensa kuntoa. Tämän ammattiryhmän perusteella työkalu sai nimen ”Doorman”.

Ohjelman kehittämisen varhaisessa vaiheessa päätettiin myös, että työkalu tulee käsittelemään nappia painamalla yhden lähdeaineiston kerrallaan. Näin haluttiin sekä helpottaa ohjelman kehitystyötä että yksinkertaistaa mahdollisien virhetilanteiden hallinnointia. Jotta pystyttiin takaamaan, että alkuperäinen lähdeaineisto ei muutu millään tavalla ohjelman toiminnoista johtuen, toimintamalliksi valittiin lähdeaineiston kopiointi työkaluun. Ajatus tässä mallissa on, että ohjelma avaa käsiteltävän lähdeaineiston työkirjan, kopioi tämän sisällön itseensä ja lopuksi sulkee käytetyn työkirjan. Kaikki muokkaustoiminnot tehdään lähdeaineiston kopiolle vasta kopioinnin jälkeen. Näin toimimalla alkuperäinen data on turvassa, vaikka tiedon käsittelyssä tapahtuisi virheitä.

3.4 Kehitetyn VBA-ohjelman toimintaperiaate

Doorman-työkalussa on jokaista KPI-seurannassa tarvittavaa Excel-työkirjaa kohden oma välilehti, jolle kyseisen työkirjan sisältö kopioidaan ja jossa se tarkastetaan, muokataan haluttuun muotoon ja johon se lopuksi myös tallentuu. Kun tarvittavat ja oikeellisiksi todetut tiedot löytyvät Doormanista, imaistaan tiedot täältä KPI-raportointijärjestelmästä käynnistettävällä VBA-ohjelmalla, joka myös kehitettiin opinnäytetyön yhteydessä. Tämän toisen ohjelman toimintaperiaate on lähes identtinen Doormanin kanssa ja siksi sen toiminnallisuutta ei esitellä raportissa erikseen.

Tarkistustoiminnon kehittäminen kaikille seurannassa mukana oleville aineistoille oli välttämätöntä, koska toimeksiantajan MIS:stä ajettujen raporttien otsikkotiedot muuttuvat silloin, kun raportille tulostuu otsikkotietue, jolla ei ole ollut aiemmin aktiivisuutta. Tämän takia jonkin kuukauden kohdalla voi olla MIS-raportilla oikea määrä riviotsikoita, mutta nämä saattavat erota KPI-raportointijärjestelmässä olevista otsikoista. Ilman otsikoille suoritettavaa tarkistusta olisi seuraava skenaario mahdollinen: KPI-raportointijärjestelmässä eräälle KPI-mittarille on ylläpidetty kohdemaat Ruotsi, Norja, Tanska ja Saksa. Mikäli MIS-järjestelmästä ajetussa raportissa olisi maat Ruotsi, Norja, Puola ja Saksa, niin Puolan tiedot kohdistuisivat Tanskan riville, ja KPI-raportointijärjestelmä antaisi toimeksiantajan johtoryhmälle virheellistä tietoa.

Kaikkien Excel- sekä Notepad-tiedostojen muokkaus- ja tarkistustoiminnot suoritetaan Doormanissa kuvion 14 pääohjelmalla, joka on helppo kopioida kaikille jatkossa KPI-seurannan piiriin lisättäville lähdeaineistolle. Ylläpitäjän tarvitsee vain vaihtaa kopioituun koodiin sekä ohjelman että tähän liittyvän työkalussa olevan välilehden nimi. Näiden muutosten jälkeen koodi toimii sellaisenaan kaikille työkaluun ylläpidetyille lähdetiedostoille.

```
Sub POF_NE()
    Application.ScreenUpdating = False
    Sheets("POFNE").Select
    Doorman
End Sub
```

KUVIO 14. Doormanin kaikille lähdeaineistoille yhteinen pääohjelma

Toimeksiantaja havaitsi aivan projektin loppupuolella, että he eivät saa kaikkia tarvitsemiaan lähdeaineistoja Excel-työkirjoina, vaan osa näistä saadaan ainoastaan Notepad-tiedostoina. Tämän takia näistä tiedostoista ei ollut mainintaa vaatimusmäärittelyissä. Suunnittelussakaan ei erikseen mainittu näitä Notepad-tiedostoja, sillä näiden koettiin olevan ainoastaan lähdeaineistoja, joiden muotoilu poikkesi aiemmista lähdeaineistoista.

Pääohjelman ensimmäinen rivi estää Exceliä päivittämästä kuvaruutua ohjelman suorittamisen aikana. Tämä nopeuttaa ohjelman suorittamista ja lisää työkalun käyttömukavuutta kun näyttö ei vilku ohjelman suorittamisen aikana häiritsevästi. Toisella rivillä kerrotaan ohjelmalle sen välilehden nimi, jolle muokattava data tullaan kopioimaan lähdetiedostosta ja jossa kaikki tarvittavat muokkaus- ja tarkastustoiminnot tullaan suorit-

tamaan. Ohjelma löytää välilehden nimen perusteella myöhemmin ohjelman suorittamisen aikana sekä lähdeaineiston sisältävän tiedoston nimen että hakemistopolun kohdekansioon, johon kyseinen tiedosto on tallennettu.

Viimeinen pääohjelman rivi käynnistää aliohjelman Doorman, joka sisältää kaikki tarvittavat toiminnallisuudet, jotka ohjelman tulee tehdä lähdeaineistoille. Kuviossa 15 on esitelty ohjaustaulun rakenne kahden lähdedatan osalta. Doormanin toimintaprosessin kuvaus on esitelty raportin osiossa liitteet.

A	B	C	D	E
x				
LogisticsDeliveriesLand	Logistics volumes trucks.xlsx	G:\KPI data\Transport		
LogisticsDeliveriesLandSubject	as values	A4		
LogisticsDeliveriesLandTitleRows&Columns	1	1		
LogisticsDeliveriesLandData	Estonia	Finland	Norway	Sweden
DelLogisticsDeliveriesLand	(Blank)			
XYLogisticsDeliveriesLand	1	0	row	
StatusLogisticsDeliveriesLand	OK			
LogisticsDeliveriesLandExtra	Sub 50	Sub 110	Sub 120	Sub 130
x				
Stocks	Stock report.txt	G:\KPI data\Stock		
StocksSubject				
StocksTitleRows&Columns				
StocksData				
DelStocks				
XYStocks				
StatusStocks				
StocksExtra	Sub 10			
x				

KUVIO 15. Doorman-ohjelmassa suoritettavia toimintoja hallinnoiva ohjaustaulu

Ohjaustauluun on erotettu A-sarakkeessa olevilla x-merkeillä eri lähdeaineistoja koskevat määritykset toisistaan. X-merkkien avulla ylläpitäjä havaitsee lähdeaineiston ohjaustietojen rajat, sillä taulun A-sarakkeessa ei saa olla tyhjiä soluja. Mikäli tällaisia olisi, ohjelma tulkitsisi käyneensä koko ohjaustaulun läpi eikä välttämättä löytäisi hakemaansa tietoa. Jokaista lähdeaineistoa varten on ohjaustauluun ylläpidetty vähintään 8 riviä ohjaustietoja, vaikka kaikki rivit eivät ole kaikille aineistoille merkityksellisiä. Niille aineistoille on ylläpidetty ohjaustauluun 9 riviä, joille lisätään ohjelman käsittelyn aikana ylimääräinen otsikkosarake.

Ohjaustaulun toiminta-ajatus on, että ohjelma käy tarkistamassa tästä taulusta ohjelman suorittamisen edetessä, miten sen tulee toimia missäkin tilanteessa. Ohjelma tunnistaa hakemansa rivin A-sarakkeessa olevista riviotsikoiden nimistä: näissä esiintyy kohde-

tiedoston määrityksistä kertovaa riviä lukuun ottamatta sekä välilehden nimi että tunniste, joka kertoo sekä ylläpitäjälle että ohjelmalle, mitä tietoa kyseisellä rivillä on. Kohdetiedosto löytyy riviltä, jonka riviotsikko on pelkkä välilehden nimi.

Riville, jonka riviotsikko on käsiteltävän välilehden nimi + Data, on ylläpidetty otsikot, joiden tulee esiintyä lähdeaineistossa täysin samassa järjestyksessä huolimatta siitä, tarkistetaanko sarake- vai riviotsikoita. Tämän tiedon ohjelma saa riviltä, jonka otsikko on XY + käsiteltävän välilehden nimi. Riville, jonka otsikko on käsiteltävän välilehden nimi + Extra, on ylläpidetty niiden aliohjelmien tunnukset, jotka ohjelman tulee suorittaa kyseiselle lähdeaineistolle. Mikäli ohjaustiedot on ylläpidetty virheellisesti, on ohjelmalla riski ajautua virheeseen. Tai mikä pahinta, ohjelma voi suorittaa koodin määritysten mukaisesti ilman järjestelmävirheitä, mutta tehden toimintoja joita se ei joko saisi tehdä tai jättää näitä tekemättä. Esimerkiksi sarakeotsikoiden tarkistamisen tekemättä jättäminen mahdollistaisi ohjelman suorittamisen jatkumisen, vaikka lähdeaineisto eroaisi asetuksiin ylläpidetystä, eikä käyttäjä saisi tästä tietoa.

Muita ohjaustaulussa ylläpidettyjä tietoja ovat lähdeaineistojen otsikkorivien- ja sarakkeiden määrät, poistettavien otsikoiden nimet, koordinaatit, joilla ohjelma saadaan tekemään tarkastuksia haluttuun suuntaan sekä tieto siitä, ovatko kyseisen lähdeaineiston otsikot riveillä vai sarakkeilla. Doorman ylläpitää lähdeaineiston statustiedon tähän tauluun aina kun lähdedatan käsittely ja tarkistus on suoritettu onnistuneesti. Tämä tieto kertoo toiselle opinnäytetyön yhteydessä kehitetylle ohjelmalle kyseisen lähdeaineiston olevan 100 % oikeellista ja näin mahdollistaen tietojen siirtämisen varsinaiseen KPI-raportointijärjestelmään.

Kaikissa Doormanin apuohjelmissa on mahdollisten virhetilanteiden varalta tarkastustoiminto, joka mahdollistaa ohjelman hallitun pysähtymisen ehkäisten virheellisen datan syntymistä sekä ohjelman jumiutumista ikuisiin silmukoihin. Virheen sattuessa käyttäjälle tiedotetaan kuvaavia virheilmoituksia tällä toiminnallisuudella, ohjelman muuntaessa virheilmoituksen monivalintalauseen avulla selkokieliseksi tekstiksi. Kuviossa 16 esitellään tilanne, jossa ohjelmaa käsketään suorittamaan toimintoja silmukassa, kunnes vastaan tulee tyhjä solu.

```
    Do Until ActiveCell.Value = ""
        If ActiveCell.Value = pName & "Extra" Then
            GoTo jump 1
        End If
        ActiveCell.Offset(1, 0).Select
    Loop
    get111 = "Error 170" 2
    GoTo stopthis

jump:
    Do Until ActiveCell.Value = ""
```

KUVIO 16. Aliohjelmissa käytettyjen silmukoiden toimintaperiaate

Silmukan toimintaperiaate on, että ohjelma etenee jokaisella pyörähdyksellä yhden rivin alaspäin tarkastaen aktiivisen solun arvon. Mikäli aktiivisessa solussa on dataa, mutta tämä ei ole haluttu tieto, siirtyy ohjelma rivin alaspäin ja silmukan suorittaminen palautuu jälleen alkuun. Kun ohjelma saapuu soluun, jonka sisältö vastaa haettua arvoa, ohjelma poistuu silmukasta hyppäämällä koodin kohtaan *jump* (merkki 1 kuviossa 16), jatkaen tästä ohjelman suorittamista.

Jos vastaan tulee tyhjä solu, silmukan suorittaminen päättyy ja ohjelma saapuu koodissa silmukan jälkeiselle riville, jonka yli ohjelma hyppäisi haettavan arvon löytyessä ohjaustaulusta. Nyt aliohjelman nimen mukaan nimetty muuttuja saa arvokseen yksilöllisen virhenumeron (merkki 2 kuviossa 16). Tämän jälkeen ohjelma hyppää *GoTo stopthis* -komennolla suoritettavan aliohjelman loppuun jättäen välistä kaikki toiminnallisuudet. Tällä tavalla ohjelmaa estetään suorittamasta normaaleja toimintoja, sillä muuten ohjelma ajautuisi väistämättä hallitsemattomaan virhetilanteeseen.

Seuraavassa esimerkissä on kuvattu virhetarkistus, joka tapahtuu jokaisen aliohjelman suorittamisen jälkeen ja jossa tarkastetaan onko aliohjelman suorituksessa mahdollisesti tapahtunut virhetilanteita.

```
myP = get40(n)
errTest = Left(myP, 5)
If errTest = "Error" Then
    GoTo errorhandler
End If
```

Edellisessä esimerkissä muuttuja *errTest* saa arvokseen muuttujan *myP* viisi vasemman puoleisinta merkkiä. Mikäli aliohjelmassa on tapahtunut virhetilanne, nämä viisi merkkiä muodostavat tekstin *Error* ja tällöin seuraavan IF-lauseen myötä ohjelman suorittaminen siirtyy kohtaan *errorhandler*, jossa käyttäjälle annetaan kyseistä virhettä kuvaava selkokielinen virheilmoitus ja ohjelman suorittaminen myös päättyy hallitusti. Vastaavasti mikäli muuttujan *errTest* arvoksi tulee mikä muu tahansa kuin *Error*, ohjelman suorittaminen jatkuu suunnitellusti.

3.5 Projektin tavoitteiden saavuttaminen

Opinnäytetyön tärkein tavoite, eli KPI-seurantaa aiemmin hoitaneiden henkilöiden vapauttaminen todellista asiantuntijuutta vaativien tehtävien pariin, saavutettiin ja toimeksiantaja on antanut kehitetystä työkalusta hyvää palautetta. Ennen opinnäytetyöprojektia toimeksiantajan kuukausittaiseen KPI-seurantaan kului 1,5 - 2,5 työpäivää, mutta nykyisin seurannassa tarvittavien työvaiheiden suorittamiseen kestää enää hieman alle 4 tuntia. Nykyisessä mallissa työ tehdään rauhallisesti ja tarkastaen aineisto silmämääräisesti, jotta palvelimelle siirrettävien lukujen kokoluokka vastaa varmasti aiempien kuukausien lukuja. Kaikki tarvittavat työvaiheet olisi mahdollista tehdä noin 3,5 tunnissa, mutta tällöin tarkastus jäisi tekemättä, eikä tästä puolen tunnin aikaerosta ole vastaavaa hyötyä. Kuviossa 17 on kuvattu nykyisin työhön kuluva aika työvaiheineen.

Lähdeainestoille tehtävät toiminnot	**Työn kesto**
49 Excel-tiedoston ajo MIS yms. järjestelmistä tallennuskansioihin	2 h 30 min
49 Excel-tiedostojen sekä kahden tekstitiedoston avaaminen, tiedostojen sisällön tarkistaminen ja käsittely Doorman-ohjelmalla	4 min 10 sek
Ohjaustietojen ja muiden tarvittavien muokkausten teko, mikäli lähdeaineiston otsikot poikkeavat aiemmasta (n. 8 min / aineisto)	32 min
Aineiston siirto KPI-seurantajärjestelmään, datan silmämääräinen tarkistaminen sekä siirto palvelimelle	43 min
Yhteensä	n. 3 h 50 min

KUVIO 17. Toimeksiantajan KPI-seurantaan kuluva aika projektin päättyessä

Koska opinnäytetyö käsittelee VBA-ohjelmointia, mitattiin Doorman-ohjelman suorituksen kesto sekuntikellolla. Muut ajat on mitattu minuutin tarkkuudella, sillä nämä eivät ole työn kannalta yhtä oleellisia ja näiden kesto vaihtelee ihmisen työvauhdin mukaan. Myös Doormanin suorittaman työn kesto riippuu käsiteltävien lähdeaineistojen

määrästä, mutta yhden aineiston osuus ei nouse merkittäväksi, sillä keskimäärin yhden aineiston käsittelyyn kuluu 5,2 sekuntia. Tämä aika sisältää lähdetiedoston avaamisen, tämän sisällön kopioinnin Doormaniin, tarvittavien muokkaustoimintojen tekemisen sekä kaikkien aineistossa olevien otsikoiden vertaamisen ohjaustaulun tietoihin.

Vertailuna Doormanille haluttiin selvittää, kuinka monta lähdeaineistoa ihminen pystyy tarkistamaan samassa ajassa kuin mitä Doormanilla kestää kaikkien toimintojen suorittamiseen. Aikavertailuun ei otettu mukaan muokkaustoimintoja, sillä mikäli ihminen tekee työn, tätä toimintoa ei tarvitse tehdä. Doormanin on kuitenkin muokattava aineistot, jotta ne ovat yhdenmukaisia tarkastustoimintojen alkaessa. Lopputulos tehdylle mittaukselle oli, että testin koehenkilö ehti tarkistaa ainoastaan kolmen lähdeaineiston otsikkotietojen yhdenmukaisuuden. Vaikka kaikki kolme tiedostosta sijaitsivat samassa tallennuskansiossa, jota ei suljettu välillä, niin silti työhön kului aikaa 4 minuuttia 47 sekuntia, eli enemmän kuin Doormanilla kestää kestää 51 aineiston läpikäymisessä.

Osa käytetystä ajasta kului oikeaan kohdekansion siirtymiseen huolimatta siitä, että tähän käytettiin pikakuvaketta. Myös tiedostojen havaitsemiseen kansion tiedostoluettelosta meni oma aikansa, kuten myös tiedostojen avaamiseen. KPI-seurantajärjestelmässä tuli myös siirtyä aina oikealle lähdeaineiston välilehdelle, jonka jälkeen vasta oli mahdollista aloittaa vertailun suorittaminen KPI-seurantajärjestelmän ja MIS-järjestelmästä ajetun raportin välillä. Mikäli koehenkilön käyttämä aika suhteutetaan 51 lähdeaineiston tarkistamiseen, tulisi työn kestoksi 1 tunti 21 minuuttia. Tässä hypoteesissa ei saisi pitää yhtään taukoa eikä mikään häiriö saisi keskeyttää työskentelyä. Ihminen ei kuitenkaan pystyisi pitämään kovin pitkään yhtä nopeaa vauhtia yllä, kuin mitä testitilanteen muutaman työkirjan vertailussa oli mahdollista pitää. Todellisuudessa vertailun tehneen henkilön työvauhti olisi hiipunut, mikäli hän olisi vielä tarkistanut muutamia aineistoja, samalla kun virheriski olisi kasvanut koko ajan työn edetessä.

Doormanista saatiin kehitettyä tavoitteiden mukaisesti toimintavarma työkalu. Mikäli lähdeaineiston tallentaja on tallentanut kaikki lähdeaineistot, kuten näiden säännöt on ylläpidetty työkalun ohjaustauluun, ei virhetilanteita pääse tapahtumaan. Tämä väite perustuu sekä läpi koko ohjelman kehitysprosessin jatkuneeseen testaamiseen että ohjelman suorituksen keskeytymiseen, mikäli ohjelma havaitsee vähäistäkään poikkeavuutta annetuista säännöistä.

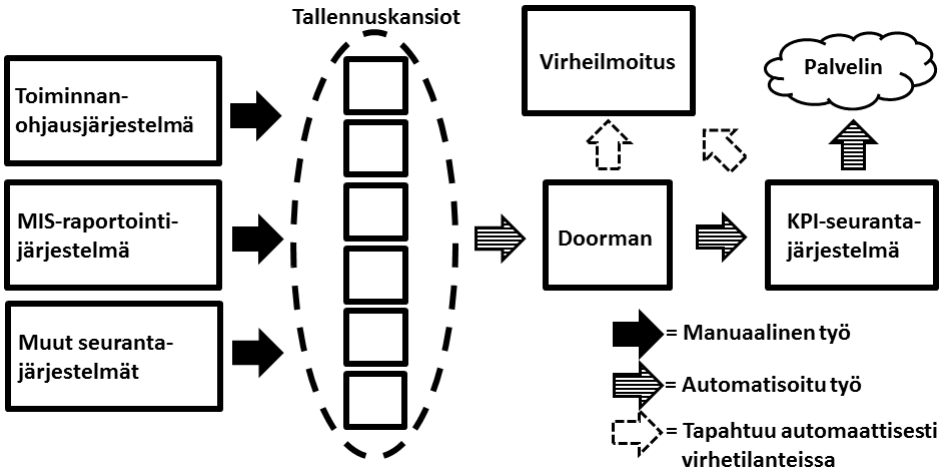
Tavoite yhdestä kaikille lähdeaineistoille yhteisestä koodista, joka oli opinnäytetyön tekijän itse asettama, onnistui yli odotuksien. Projektin alkaessa luultiin, että tarvittavaan ohjelmaan kertyisi koodia useita satoja rivejä, ja että ohjelman ylläpitäjän tulisi kopioida tämä kokonaisuus kaikille ohjelmaan lisättävälle lähdeaineistolle. Projektin aikana saatu vinkki VBA-ohjelmoinnissa hyödynnettävistä aliohjelmista kuitenkin mahdollisti pääohjelman kehittämisen, jonka pituus on ainoastaan 3 riviä koodia. Työkalun ylläpitäjän tarvitsee jatkossa kopioida ainoastaan tämä ohjelma työkaluun lisättäville lähdeaineistoille. Vielä suurempi hyöty aliohjelmien hyödyntämisestä saavutetaan kun jatkossa tulee tarve muuttaa ohjelman koodia. Tällöin riittää, kun muutoksen tekee yhteen aliohjelmaan. Jos ohjelma olisi toteutettu alun perin suunnittelulla tavalla, muutos olisi pitänyt tehdä jokaiseen työkalussa olevaan 51 ohjelman kopioon.

Osa asetetuista tavoitteista jäi toistaiseksi saavuttamatta. Koska Doormanin kehittäminen kesti KPI-mittariston kehitysprojektin takia suunniteltua kauemmin, ei työkalun helppokäyttöisyyttä ehditty testata Excelin peruskäyttäjillä. Tästä huolimatta ongelmia tuskin tulee, sillä Doormanista pystyy yhdellä napin painalluksella käynnistämään toimintasarjan, joka käsittelee kaikki ohjelmaan ylläpidetyt lähdeaineistot. Halutessaan käyttäjä voi käsitellä aineistot myös yksi kerrallaan työkalussa olevien painonappien avulla, joiden vieressä lukee selkeästi mitä lähdeaineistoa kyseinen nappi koskee. Tilattu VBA-työkalu valmistui määritettyjen tavoitteiden mukaisena ja aikataulun puitteissa, mutta tähän liittyvät englanninkieliset käyttöohjeet toimitetaan toimeksiantajalle vasta opinnäytetyöprojektin jälkeen.

Aikataulullisista syistä johtuen ei myöskään uusien lähdeaineistojen liittämistä ohjelman käsiteltäväksi ehditty testata muiden kuin ohjelman kehittäjän toimesta. Vaikka työkalun ylläpitoon liittyvät toiminnot ovat hieman peruskäyttöä monimutkaisempia, voi työn tehdä kuka tahansa Excel-käyttäjä selkeiden kirjallisten ohjeiden perusteella. Työkalussa on uusien lähdeaineistojen lisäämistä varten oma VBA-ohjelma, joka tekee lähes kaikki tarvittavat toiminnot ylläpitäjän puolesta. Tämän ohjelman käyttö kuitenkin edellyttää, että lähdeaineiston muotoilu vastaa Doormanin nykyisin tukemia muotoiluja. Tästä ehdosta tuskin muodostuu ongelmaa, sillä Doorman käsittelee nykyisin 14 erimuotoista lähdeaineistoa.

Kehitystyön aikana toimeksiantajan KPI-seurantaprosessissa tapahtuneista muutoksista johtuen ei ollut mahdollista mitata sen ajan määrää, jonka tehostuminen johtui ainoastaan kehitetystä VBA-ohjelmasta. Samasta syystä hylättiin jo hyvin aikaisessa vaiheessa projektia tavoite koko työkalun kehitykseen kuluvan ajan mittaamisesta. Mittaaminen lopetettiin, koska projektinhallinnallisista syistä tällaista ohjelmakehitystä ei kannata tehdä koko ajan muuttuvassa ympäristössä.

Toimeksiantajan nykyisessä KPI-seurantaprosessissa, joka on esitelty kuviossa 18, tallennetaan kaikki lähdeaineistot yrityksen verkkoasemalla sijaitseviin tallennuskansioihin. Doorman avaa lähdeaineistojen tiedostot, kopioi näiden sisällön itseensä ja sulkee lähdetiedostot. Kopioitujen aineistojen muotoilut muutetaan yhdenmukaisiksi seuraavaksi suoritettavaa tarkistustoimintoa varten. Kun kaikki tiedostot on käsitelty, tästä annetaan käyttäjälle viesti sanomaikkunan välityksellä ja käyttäjä voi tallettaa työkaluun tehdyt muutokset. Tämän jälkeen käyttäjä avaa varsinaisen KPI-seurantajärjestelmän ja käynnistää täältä ohjelman, joka imaisee tallennetut tiedot Doormanista. Lopuksi käyttäjä tarkistaa KPI-seurantajärjestelmään siirtyneet tiedot silmämääräisesti samalla kun hän käynnistää ohjelman, joka syöttää KPI-luvut palvelimelle. Mikäli Doormanin tai KPI-seurantajärjestelmästä käynnistetyn ohjelman suorituksessa tapahtuu virhetilanne, käyttäjä saa tästä kuvaavan virheilmoituksen ja ohjelman suoritus loppuu hallitusti.



KUVIO 18. Kuvaus toimeksiantajan nykyisestä KPI-seurantaprosessista

Vaikka toimeksiantajan KPI-mittausprosessi muuttui projektin aikana, tästä huolimatta opinnäytetyön aikana suoritetuilla mittauksilla pystytään kiistattomasti osoittamaan VBA-ohjelmoinnin hyöty niin toimeksiantajalle tämän KPI-mittausprosessin tehostumisessa kuten myös yleisemmin tarkasteltuna.

4 YHTEENVETO

4.1 Opinnäytetyö yleisesti

Opinnäytetyölle yleisesti asetetut tavoitteet onnistuttiin täyttämään. VBA:n perustoiminnallisuuksia esiteltiin riittävän laajasti menemättä kuitenkaan täysin ruohonjuuritasolle, tarkoitushan oli ainoastaan esitellä tämän ohjelmointitavan tarjoamia mahdollisuuksia, ei tehdä käyttöopasta tästä ohjelmointikielestä. Myös esiteltyjen käyttötapausesimerkkien avulla esimies- ja asiantuntijatehtävissä olevat henkilöt pystyvät tunnistamaan sellaiset Excelillä tehtävät työt, jotka voi automatisoida turhan ja tuottamattoman työn vähentämiseksi sekä työviihtyvyyden parantamiseksi. Tietoturvariskit käytiin läpi ja annettiin myös ohjeistus siitä, kuinka jokainen Excelin käyttäjä voi tarkistaa oman työasemansa Excelin makrojen tietoturva-asetukset ja tarvittaessa muuttaa nämä suositusten mukaisiksi.

4.2 Toimeksiantajalle kehitetty VBA-ohjelma

Opinnäytetyö vastaa lähes kaikkiin kysymyksiin, joihin sen tulee vastata. Periaatteessa kaikki muut Excelillä suoritettavat toimeksiantajan KPI-seurantaan liittyvät prosessit on automatisoitu kehitetyn VBA-ohjelman avulla paitsi muutamat ylläpitotehtävät, joita työkalun ylläpitäjän tulee tehdä lisätessään työkalun käsiteltäväksi uusia lähdeaineistoja. Koska tätäkin tehtävää varten on kehitetty oman VBA-ohjelma, niin käytännössä ylläpitäjän tulee ainoastaan tietää työkirjaan lisättävän välilehden nimi, kuten myös lähdeaineiston tiedostonimi sekä tämän tallennuskansio.

Teknisesti olisi ollut mahdollista automatisoida myös tarkastettujen ja muokattujen tietojen syöttö varsinaisen KPI-seurantajärjestelmän palvelimelle, mutta tämä toiminto haluttiin jättää automatisoimatta. Päätös tehtiin siksi, koska ohjelman käyttäjä pystyy suorittamaan tämän toiminnon järjestelmästä nappia painamalla, niin hänet haluttiin pakottaa toteamaan silmämääräisesti aineiston vastaavan sisällöltään aiemmin tallennettua. Mikäli poikkeamia esiintyisi, käyttäjä pystyisi tässä vaiheessa tekemään tarvittavat tarkistustoimenpiteet tai ainakin tiedottamaan asiasta edelleen.

Kehitetty Doorman-ohjelma on koekäytön perusteella Excelin peruskäyttäjiä ajatellen riittävän skaalautuva. Jotta Excelin peruskäyttäjät voivat toimia jatkossa niin työkalun normaaleina käyttäjinä kuin myös tämän ylläpitäjinä, on työkalusta ensin kirjoitettava selkeät englanninkieliset käyttöohjeet. Syy käytettävälle kielelle on toimeksiantajan globaalit toiminnot, sillä suomenkielisillä ohjeilla ei muiden maiden toimipisteissä työskentelevät henkilöt tekisi käytännössä mitään. Jo tällä hetkellä peruskäyttäjiltä onnistuisi työkalun normaali käyttö lyhyen suullisen ohjeistuksen perusteella, mutta ylläpitojen lisääminen tai näiden muuttaminen työkaluun olisi heille ilman kirjallisia ohjeita liian haasteellista.

Projektin edetessä osa saadusta aineistosta todettiin toimeksiantajan taholta sellaiseksi, jota ei enää jatkossa tarvittu, mutta näiden tilalle tuli paljon uutta aiemmin käyttämätöntä aineistoa. Suurin osa uudesta aineistosta oli muotoiluiltaan identtistä poistettavan aineiston kanssa, joten kovin paljoa ei jouduttu kehittämään koodia, joka osoittautui vanhentuneeksi jo ennen projektin valmistumista. Tästä huolimatta myös projektin ulkopuolelle jääneet aineistot veivät työaikaa, sillä myös näitä varten jouduttiin tekemään sekä Doormaniin että toimeksiantajan varsinaiseen KPI-seurantatyökaluun tarvittavia asetuksia. Oman aikansa vei myös jokaisen yksittäisen lähdeaineiston käsittelyn testaaminen Doormanilla.

Toimeksiantajan toimittaessa uusia lähdeaineistoja, joiden muotoilu poikkesi aiemmin käytetyistä, toi varsin paljon lisähaastetta tarvittavan koodin suunnitteluun. Paitsi että näitä varten tuli kehittää uusi toiminnallisuus, niin poikkeuksetta myös jo olemassa olevaa koodia oli muutettava. Oman paineensa koodin suunnitteluun toi myös halu kehittää ohjelma hyvien ohjelmointiperiaatteiden mukaisesti, välttäen ylimääräisiä koodirivejä ja samojen toiminnallisuuksien toistoa eri aliohjelmissa.

Projektin alkuvaiheessa pidettiin tekemättömistä tehtävistä listaa ruutuvihkoon, josta näitä yliviivattiin sitä mukaa kun ne valmistuivat. Vasta projektin loppupuolella alettiin pitää tehtävistä Excel-tiedostoa, joka mahdollisti tehtävien priorisoinnin viisiportaisella järjestelmällä tehtävien kiireellisyyden perusteella. Tämä taulukko olisi pitänyt olla käytössäni heti projektin alusta alkaen, sillä tämän avulla oli helpompi sekä hallinnoida suoritettavia tehtäviä että suorittaa samaan aiheeseen liittyviä tehtäviä samanaikaisesti.

Doormanin kehitystyö olisi ollut nopeampaa, mikäli sen lähtökohtana ei olisi ollut mahdollisimman korkea käyttäjäystävällisyys ja skaalautuvuus. Kaikki ohjelman nykyisin suorittamat toiminnot olisi ollut mahdollista kehittää muutaman työpäivän aikana, mikäli kullekin lähdeaineistolle olisi tehty erillinen VBA-ohjelma. Tällöin myös ohjelmien toiminnallisuudet olisi kirjoitettu suoraan näiden ohjelmien koodiin, ilman nyt käytettyjä muuttujia. Näin ei kuitenkaan ollut mahdollista toimia tämän kehitystyön yhteydessä, sillä tämä olisi estänyt Excelin peruskäyttäjien toimimisen jatkossa ohjelman ylläpitäjinä. Ongelma ratkaistiin käyttämällä samaa logiikkaa kuin mitä toimeksiantajan yhteistyökumppani oli käyttänyt omassa Excel-työkalussaan. Tämä ratkaisumalli on ohjaustaulu, johon ylläpidetään ohjaustiedot ylläpitäjän toimesta kuten mihin tahansa Excel-tiedostoon ja ohjelma käy täältä tarkistamassa koodin suorittamisen aikana kaikki lähdeaineistoille tehtävät toiminnallisuudet.

Opinnäytetyön yhteydessä kehitetyn Doormanin kehitystyöhön käytetyn ajan mittaaminen lopetettiin kesken, sillä mittaustulos ei olisi ollut projektin olosuhteista johtuen verrannollinen normaalisti VBA-ohjelmien kehitystyöhön käytettävään aikaan. Mikäli kehitystyö olisi ollut mahdollista suorittaa olosuhteissa joissa käytettävä lähdeaineisto on etukäteen tiedossa, varsinaiseen ohjelmointityöhön olisi sen kehittäjältä kulunut arviolta noin 2 – 2,5 työviikkoa. Tämän lisäksi sekä ennen ohjelmointityön aloittamista että ohjelmoinnin aikana tehtävään suunnittelutyöhön olisi tullut varata noin viikko. Kokonaistyöaikaa näin laajan työkalun kehitykseen tulee siis varata testaukset mukaan lukien arviolta 3 – 3,5 viikkoa, joka on työpäiviksi muutettuna noin 18 päivää. Vertailuna tälle arviolle voidaan käyttää toimeksiantajan aiemmin keskimäärin kaksi työpäivää kuukaudessa vienyttä KPI-seurannan toteuttamista, joka on tehnyt noin 18 työpäivää yhdeksän kuukauden aikana. Mikäli arvioidaan kehitystyön hinta ja toimeksiantajan KPI-seurannan toteuttamisen hinta samanarvoisiksi, tulisi tehty kehitystyö maksamaan itsensä takaisin 9 kuukaudessa.

4.3 VBA-ohjelman jatkokehitys

Doorman täyttää projektin päättyessä toimeksiantajan sille asettamat tekniset vaatimukset. Tästä huolimatta työkalu vaatii vielä jatkokehitystä, jotta se täyttää myös työkalun kehittäjän omalle työlleen asettamat laatuvaatimukset. Seuraavalle listalle on koottu

sellaisia ehdotuksia jatkokehityksen kohteiksi, joiden avulla Doormanista on mahdollista kehittää vielä nykyistäkin käyttäjäystävällisempi ja skaalautuvampi työkalu. Näillä toimilla on mahdollista sekä tehostaa toimeksiantajan työajankäyttöä että yksinkertaistaa KPI-seurantaprosessia vielä hieman verrattuna nykytilaan.

- Työkaluun on lisättävä tarkistusohjelma, jota käyttämällä ylläpitäjä voi selvittää automaattisesti, miltä työkalussa jo olevalta lähdeaineistolta on mahdollista kopioida ylläpidot uudelle työkaluun lisättävälle lähdeaineistolle.
- Tekstitiedostoja avaavaan aliohjelmaan on lisättävä muuttujat, jotka mahdollistavat lähdetiedostojen avaamisen ainoastaan yhtä aliohjelmaa käyttäen huolimatta avattavan tiedoton nimestä ja tallennuskansiosta. Nykyisin työkaluun ylläpidetyille molemmille tekstitiedostolle on oma aliohjelma, sillä nämä lisättiin työkaluun vasta aivan projektin lopussa.
- Tarvetta on jatkossa myös ohjelmalle, jonka avulla ylläpitäjä voi tehdä muutokset Doormanin ohjaustauluun tilanteissa, jolloin lähdeaineistossa on tapahtunut muutoksia. Esimerkiksi lähdeaineiston otsikkotiedon muuttuessa työkalun ylläpitäjän tarvitsisi ainoastaan ilmoittaa kyseisen välilehden nimi, jolloin ohjelma päivittäisi ohjaustaulun tiedot vastaamaan lähdeaineiston nykyisiä otsikkotietoja.
- Kun luetellut toiminnot on lisätty Doormaniin, tästä työkalusta on tarkoitus tehdä lopulta työtön. Tämä on tarkoitus toteuttaa liittämällä työkalussa käytettävät ohjelmat osaksi toimeksiantajan varsinaista KPI-seurantajärjestelmää. Näin toimimalla KPI-seurantaprosessi kevenee vielä entisestään yhden välivaiheen poistuessa.

LÄHTEET

Merensalmi, Jussi 2007. Excel VBA yrityskäytössä. Jyväskylä: WSOY.

Shepherd, Richard 2006. Excel-ohjelmointi - Tehokas hallinta. Jyväskylä: Readme.fi.

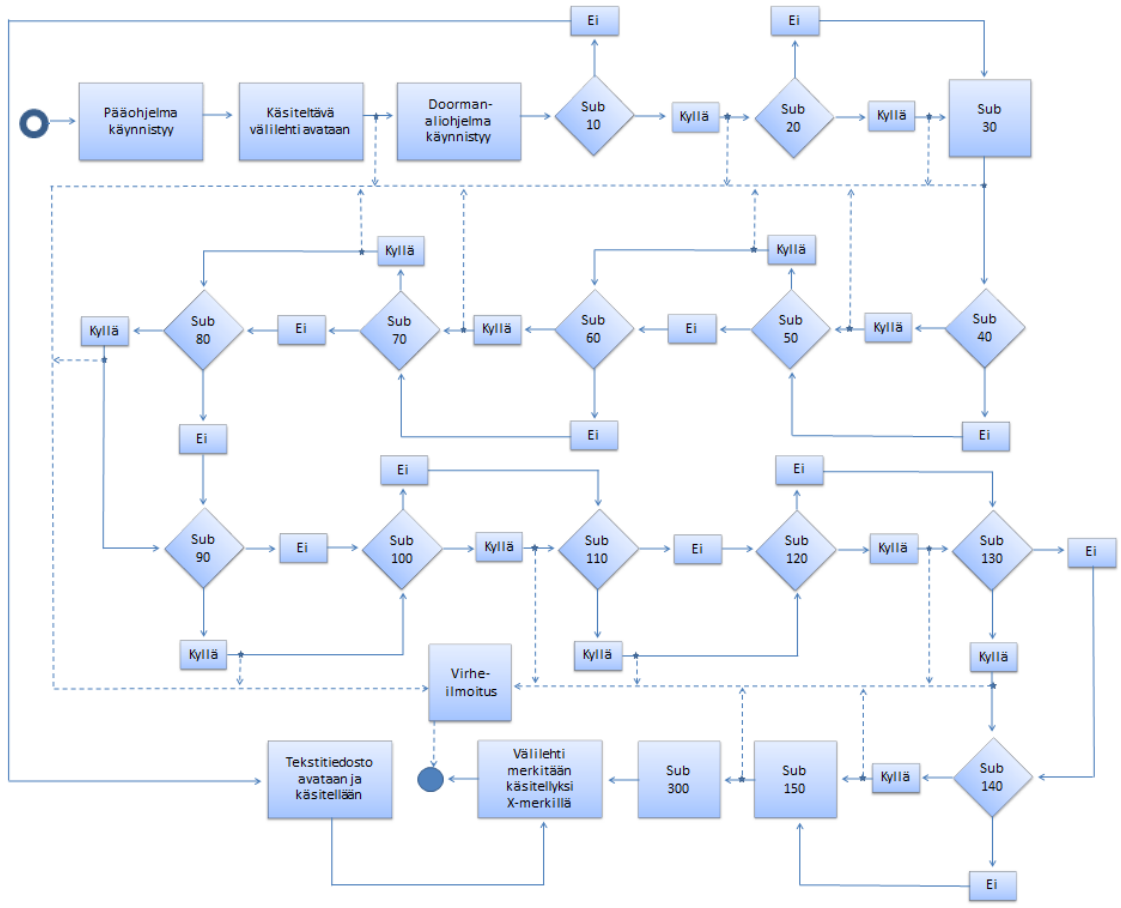
Lehto, Tero 2009. Korjaamattomaan Excel-aukkoon hyökätään. Päivitetty 24.2.2009. Luettu 4.3.2012.
http://www.tietokone.fi/uutiset/2009/korjaamattomaan_excel_aukkoon_hyokataan

Karvonen, Tuomas 2003. MS Officessa vakava haava. Päivitetty 4.9.2003. Luettu 4.3.2012.
http://www.itviikko.fi/muu/2003/09/04/ms-officessa-vakava-haava/20032709/7

http://office.microsoft.com/fi-fi/excel-help/makrojen-ottaminen-kayttoon-ja-poistaminen-kaytosta-office-tiedostoissa-HA010354316.aspx?CTT=1

LIITTEET

Doorman-ohjelman prosessikuvaus



Käytettyjen merkkien selitykset: