Markus Varila

KOOSTEOPINNÄYTETYÖ. DIGITAALINEN SIGAANLINKÄSITTELY JA OHJELMISTOTESTAUS.

KOOSTEOPINNÄYTETYÖ. DIGITAALINEN SIGAANLINKÄSITTELY JA OHJELMISTOTESTAUS.

Markus Varila
Opinnäytetyö
Kevät 2017
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, Langattomat laitteet

Tekijä(t): Markus Varila
Opinnäytetyön nimi: Koosteopinnäytetyö. Digitaalinen signaalinkäsittely ja ohjelmistotestaus.
Työn ohjaaja(t): Kari Jyrkkä, Harri Maatela, Lazo Eric
Työn valmistumislukukausi ja -vuosi: Kevät 2017 Sivumäärä: 80

Tämä koosteopinnäytetyö koostuu kahdesta osasta, jotka ovat esitetty tässä dokumentissa kahtena liitteenä.

Ensimmäisen opinnäytetyön osan tarkoituksena oli tutkia digitaalisen signaalinkäsittelyn mahdollisuuksia erityisesti audiosovelluksissa. Työssä käydään läpi yleisimpiä digitaalisen signaalinkäsittelyn tapoja ja niiden toteuttamista ohjelmallisesti. Työssä keskitytään erityisesti musiikin käsittelyyn.

Musiikin käsittely digitaalisesti luo monia ongelmia verrattuna pelkän puheen käsittelyyn. Musiikki sisältää laajan alan erilaisia äänentaajuuksia, mikä tekee signaalinkäsittelystä haastavaa. Opinnäytetyön ensimmäisessä osassa käydään läpi eri tapoja käsitellä digitaalista signaalia ja niissä huomioitavia asioita. Työssä perehdytään myös mp3-tiedostojen rakenteeseen.

Opinnäytetyön toinen osa tehtiin Nokia Oyj:ssä, jonka tarkoituksena oli suoraviivaistaa ohjelmistotestausta tiimissä, jossa työskentelin. Ohjelmistotestaukseen liittyvät päivittäiset ongelmat tunnistettiin ja niihin toteutettiin ratkaisuja.

Ongelmiin toteutettiin ratkaisut tekemällä Python-skriptejä, jotka vähentävät manuaalisen työn määrää päivittäisessä työssä.

Asiasanat: ohjelmistotestaus, testausautomaatio, tukiasema, Python, digitaalinen signaalinkäsittely

SISÄLLYS

1 JOHDANTO

Opinnäytetyö tehtiin kahdessa osassa koosteopinnäytetyönä. Ensimmäinen osa tehtiin tutkinnon toisena opiskeluvuonna ja jälkimmäinen osa tehtiin tutkinnon neljännen vuoden aikana. Ensimmäinen osa tehtiin koululla omasta aiheesta, jonka tarkoituksena oli tottua tiedonhakuun sekä lähteiden käyttöön. Toinen osa tehtiin Nokia Oyj:ssä yrityksen tarjoamasta aiheesta.

Molemmat osat on esitetty liitteinä tässä dokumentissa.

2 OPINNÄYTETYÖN OSAN 1 AIHE, TAVOITE JA TULOKSET

Opinnäytetyön ensimmäisen osan (liite 1) aiheena oli digitaalinen signaalinkäsittely ja äänen toistaminen. Opinnäytetyössä tutkittiin digitaalisen signaalinkäsittelyn mahdollisuuksia erityisesti audiosovelluksissa.

3 OPINNÄYTETYÖN OSAN 2 + 3 AIHE, TAVOITE JA TULOKSET

Opinnäytetyön osat 2 ja 3 (liite 2) toteutettiin Nokia Oyj:n tarjoamasta aiheesta. Opinnäytetyö oli luonnollinen jatkumo työlle, jota olin tehnyt samassa tiimissä kuluneen vuoden aikana projektitöiden ja kesätöiden merkeissä.

Työn tarkoituksena oli tutkia tiimissä käytetyn ohjelmistotestausautomaation ongelmia ja toteuttaa niihin ratkaisuja. Opinnäytetyössä tutustuttiin myös ohjelmistotestauksen periaatteisiin.

Työn tuloksena saatiin vähennettyä ohjelmistotestaukseen liittyviä manuaalisia toimenpiteitä.

4 YHTEENVETO

Opinnäytetyön ensimmäinen osa keskittyi aiheeseen, joka oli minulle jo entuudestaan tuttu ja mielenkiintoinen. Toinen osa keskittyi aiheeseen, joka tuli tutuksi vasta yrityslähtöisten projektitöiden sekä kesätöiden kautta. Opinnäytetyön toinen osa antoi mahdollisuuden tutustua tarkemmin ohjelmistotestausautomaatiojärjestelmiin, sekä toteuttaa niihin parannuksia.

Ensimmäinen ja toinen osa opinnäytetöistä on yhdistetty yhdeksi kokonaisuudeksi. Töiden aiheiden eroavaisuus teki opinnäytetyöstä monipuolisemman mitä tavanomaiset opinnäytetyöt.

LIITTEET

Liite 1 Opinnäytetyön osa 1: Digitaalinen signaalinkäsittelyn ja äänen toistaminen

Liite 2 Opinnäytetyön osat 2 ja 3: Improving software testing automation and preparing for cloudification

OULUN AMMATTIKORKEAKOULU

Markus Varila

DIGITAALINEN SIGNAALINKÄSITTELY JA ÄÄNEN TOISTAMINEN

DIGITAALINEN SIGNAALINKÄSITTELY JA ÄÄNEN TOISTAMINEN

Markus Varila
Opinnäytetyö, osa 1
Kevät 2015
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on tutkia digitaalisen signaalinkäsittelyn mahdollisuuksia erityisesti audiosovelluksissa. Työssä käydään läpi yleisimpiä digitaalisen signaalinkäsittelyn tapoja ja niiden toteuttamista ohjelmallisesti. Työssä keskitytään erityisesti musiikin käsittelyyn.

Musiikin käsittely digitaalisesti luo monia ongelmia verrattuna pelkän puheen käsittelyyn. Musiikki sisältää laajan alan erilaisia äänentaajuuksia, mikä tekee signaalinkäsittelystä haastavaa. Tässä työssä käydään läpi eri tapoja käsitellä digitaalista signaalia ja niissä huomioitavia asioita. Työssä perehdytään myös mp3-tiedostojen rakenteeseen.

Tämä työ on osa 15 op:n opinnäytetyötä. Tämä osa kattaa 5 op:tä koko opinnäytetyön kokonaisuudesta.

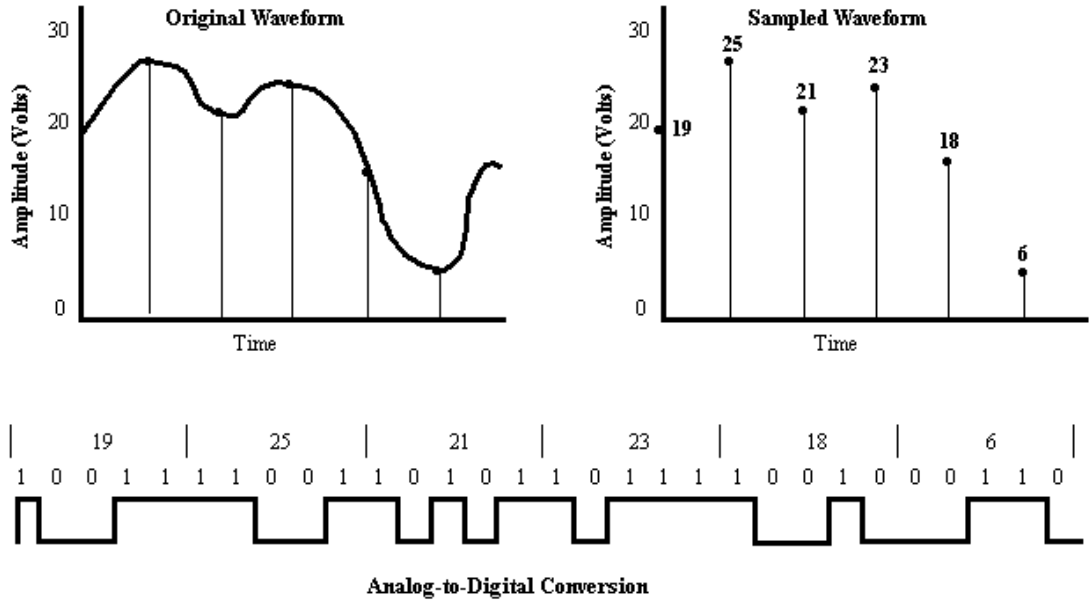
2 DIGITAALINEN SIGNAALINKÄSITTELY

Digitaalinen signaalinkäsittely(DSP, Digital Signal Processing) on tietotekniikan osa-alue, joka pyrkii kuvaamaan ja muokkaamaan signaaleja matemaattisesti käyttäen hyväksi tietokoneiden huimaa prosessointitehoa. Usein nämä signaalit ovat sensoridataa oikean elämän ilmiöistä kuten äänestä tai puheesta. (1).

2.1 Digitaalisen signaalinkäsittelyn perusteet

Useat oikean elämän ilmiöt ovat analogisia, mutta digitaalinen signaalinkäsittely nimensä perusteella käsittelee vain digitaalisia signaaleja. Tämän vuoksi analogiset ilmiöt muutetaan digitaalisiksi ADC-piireillä (Analog-to-Digital Converter). Analogiset ilmiöt ovat jatkuvia, mutta digitaalisessa muodossa oleva tieto on tallennettu vain lukuina tunnetun aikajakson välein. (1).

Kuva 1 havainnollistaa miten analoginen signaali, esimerkiksi ääni, joka on tallennettu mikrofonia käyttäen, muutetaan digitaaliseen muotoon, ja lopulta miten se voidaan esittää binaarisesti.(2).



KUVA 1. Analog-to-Digital Converter (2)

Original Waveform kuvaa analogista signaali, joka on saanut mitä tahansa arvoja välillä 0-30 volttia. Tämän jälkeen signaalista on otettu tasaisin väliajoin

näytteitä. Nämä näytteet on esitetty kuvaajassa Sampled Waveforms. Kuvaajasta näkyy, miten signaali ei ole enää analogista vaan jäljellä on enää mitattuja arvoja eri ajanhetkillä. Signaalista otetaan näytteitä tyypillisesti tasaisin väliajoin. Tätä kutsutaan näytteistykseksi. Näytteistyksen tiheyttä kutsutaan näytteistystaajuudeksi, jonka yksikkö on hertsi. Lopulta lukuarvot on esitetty 5-bittisinä binaarilukuina. Näin analoginen signaali on saatu digitaaliseen muotoon ja on valmis käsiteltäväksi.

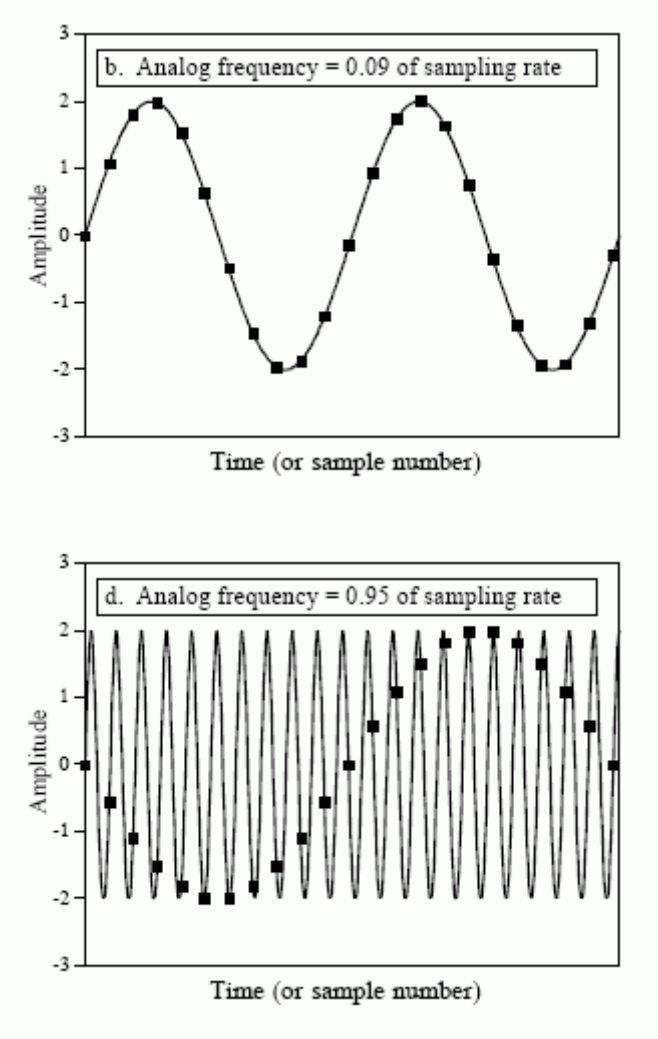
2.1.1 Näytteistys

Yksi tärkeimpiä huomioitavia asioita analogisen signaalin muuttamisesta digitaaliseksi on näytteistys. Se tarkoittaa näytteiden ottamista analogisesti signaalista tietyin väliajoin. Näytteistyksen suuruutta kutsutaan näytteistystaajuudeksi jonka yksikkö on hertsi. Esimerkiksi, jos signaalista otetaan näytteitä 100 ms:n välein, näytteistystaajuutena on 10 Hz.

Näytteistys on tärkeää sillä väärin toteutettu näytteistys ei enää sisällä kaikkea tietoa analogisesta signaalista, josta näytteistys on tehty. Oikein näytteistetty digitaalinen signaali voidaan siis palauttaa täysin samanlaiseksi analogiseksi signaaliksi.(3).

Näytteistystaajuuden tulee olla vähintään kaksi kertaa suurempi kuin signaalissa esiintyvä suurin taajuuskomponentti. Tätä taajuutta kutsutaan Nyquistin taajuudeksi. Tälläinen näytteistystaajuus takaa sen, että laskostumista ei synny. Laskostuminen tarkoittaa, että digitaalinen signaali näyttää erilaiselta kuin alkuperäinen analoginen signaali. (3; 4).

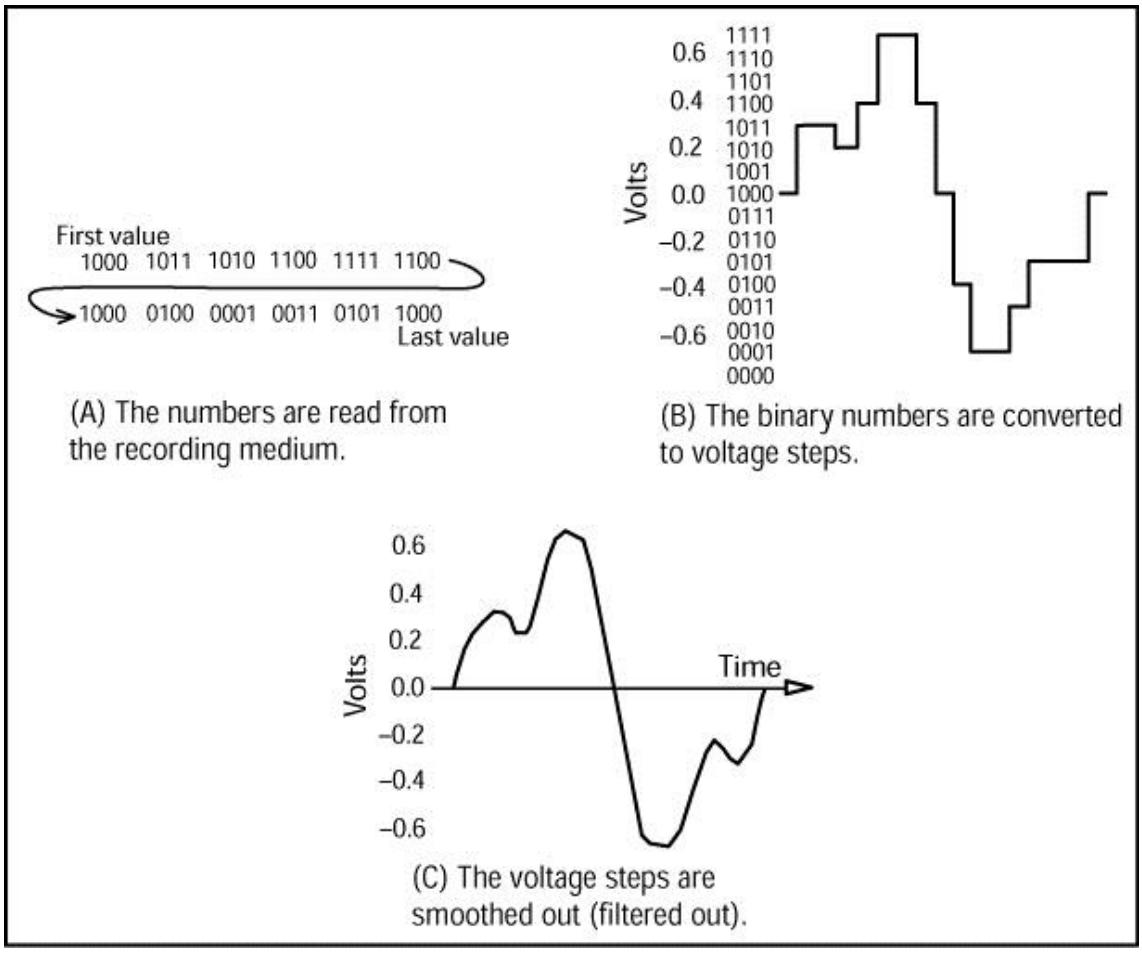
Kuvasta 2 näkyy kuinka tärkeää tarpeeksi suuri näytteistystaajuus on. Ylemmässä kuvassa näytteistystaajuus on tarpeeksi suuri ottamaan talteen kaikki vaadittava informaatio signaalista. Alemman kuvaajan signaali on vahvasti aliasoitunut. Digitaalinen signaali on selvästi pienempi taajuisempi kuin alkuperäinen analoginen signaali. Informaatio alkuperäisestä signaalista on tällöin riittämätön palauttamaan digitaalinen signaali takaisin analogiseksi.



KUVA 2. Näytteistystaajuuden tärkeys (3)

2.1.2 Digitaalisen signaalin muuttaminen analogiseksi

Digitaalinen signaali voidaan haluta muuttaa takaisin analogiseksi sen käsittelyn jälkeen. Tällöin prosessi on hyvin samantapainen kuin analogisen signaalin muuttaminen digitaaliseksi. ADC-piirin sijasta käytetään DAC-piiriä(Digital-to-Analog), joka muuttaa digitaalisen signaalin analogiseksi. Tämä ei riitä kuvaamaan analogista signaalia tarpeeksi hyvin. Tämän vuoksi DAC-piirin jälkeen signaali käytetään anti-imaging suodattimen läpi. Anti-imaging suodattimen tarkoituksena on tehdä juuri muutetusta signaalista pehmeämpi ja poistaa porrasmaisuus. Digitaalisen signaalin muuttamista analogiseksi on havainnollistettu kuvassa 3.



KUVA 3. DAC ja Anti-Imaging Suodatin (5)

2.2 Digitaalisen signaalinkäsittelyn muotoja

Digitaalisen signaalinkäsittelyn muotoja on useita. Yleisimpiä näistä ovat suodattaminen, viiveiden lisääminen ja fysikaalisten suureiden muuttaminen yksiköstä toiseen.

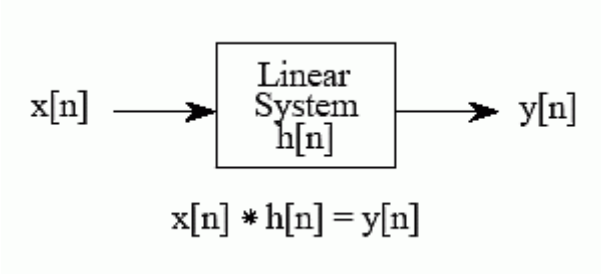
Suodattimien tarkoitus on poistaa datasta tarpeettomia taajuusalueita. Suodattimista on kerrottu tässä dokumentissa lisää myöhemmin. Viiveiden lisääminen ja säätäminen on tärkeä asia monimutkaisissa audiosovelluksissa, kuten kotiteatterijärjestelmissä. Viiveiden merkityksestä on kerrottu lisää myöhemmin tässä dokumentissa. Fysikaalisten yksiköiden muutoksia käytetään, kun data on otettu talteen jollain tiettyä suuretta mittaavalla anturilla mutta datan käsittelyssä halutaan käyttää jotain muuta tästä suureesta johdettavaa suuretta. Esimerkiksi

paikkaa mittaavalla anturilla saatu data voidaan muuttaa nopeutta kuvaavaksi dataksi. Nämä suureiden muutokset yleensä tapahtuvat derivoimalla tai integroimalla dataa ajan suhteen.

2.2.1 Konvoluutio

Konvoluutio on matemaattinen toimenpide, jossa kahta signaalia käyttäen luodaan kolmas signaali. Tämä on digitaalisen signaalinkäsittelyn perusta(1).

Kuvassa 4 on kuvattu konvoluutiota yksinkertaisesti. Funktio y(n) on funktioiden x(n) ja h(n) konvoluutio.



KUVA 4. Konvoluutio (1)

2.2.2 Fourier-muunnos

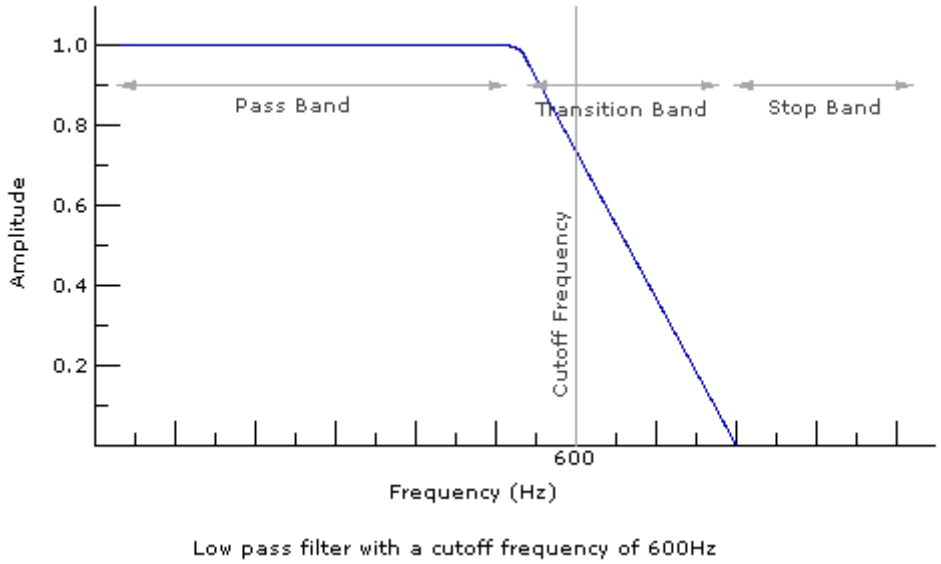
Fourier-muunnos on matemaattinen tapa esittää signaali eritaajuisilla ja –vaiheisilla sinisignaaleilla, sillä lähes kaikki signaalit voidaan esittää erilaisten sinisignaalien summana (1).

Fourier-muunnosta käytetään, jotta signaali voidaan muuttaa helpommin käsiteltävään muotoon (1).

2.2.3 Suodattimet

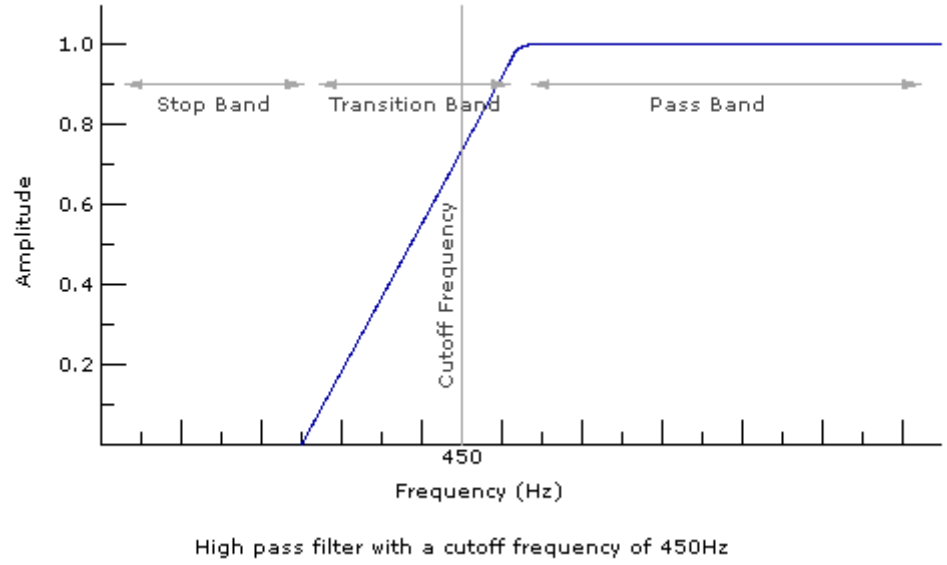
Suodattaminen on yleisimpiä signaalinkäsittelyn muotoja audiosovelluksissa. Suodattamisen tarkoitus on poistaa datasta taajuusalueita, joita ei tarvita tai jotka ovat jopa haitallisia. Erilaisia suodatintyyppejä ovat alipäästösuodatin (low pass filter, LPF) ylipäästösuodatin (high pass filter, HPF) ja kaistanpäästösuodatin (bandpass filter).

Alipäästösuodatin päästää läpi taajuudet, jotka ovat pienempiä kuin suodattimen rajataajuus (cutoff frequency). Alipäästösuodatinta on havainnollistettu kuvassa 5.



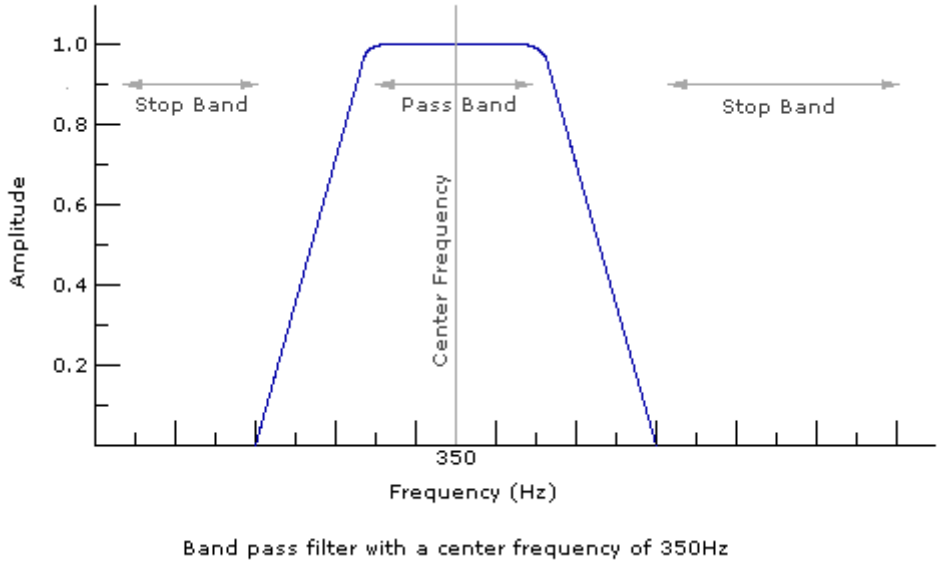
KUVA 5. Alipäästösuodatin (6)

Ylipäästösuodatin päästää läpi taajuudet, jotka ovat suurempia kuin suodattimen rajataajuus. Ylipäästösuodatinta on havainnollistettu kuvassa 6.



KUVA 6. Ylipäästösuodatin (6)

Kaistanpäästösuodatin on kuin yhdistelmä ali- ja ylipäästösuodattimia (6). Se päästää läpi vain taajuudet tietyltä taajuuskaistalta ja estää tätä kaistaa suuremmat sekä pienemmät taajuudet. Kaistanpäästösuodatinta on havainnollistettu kuvassa 7.

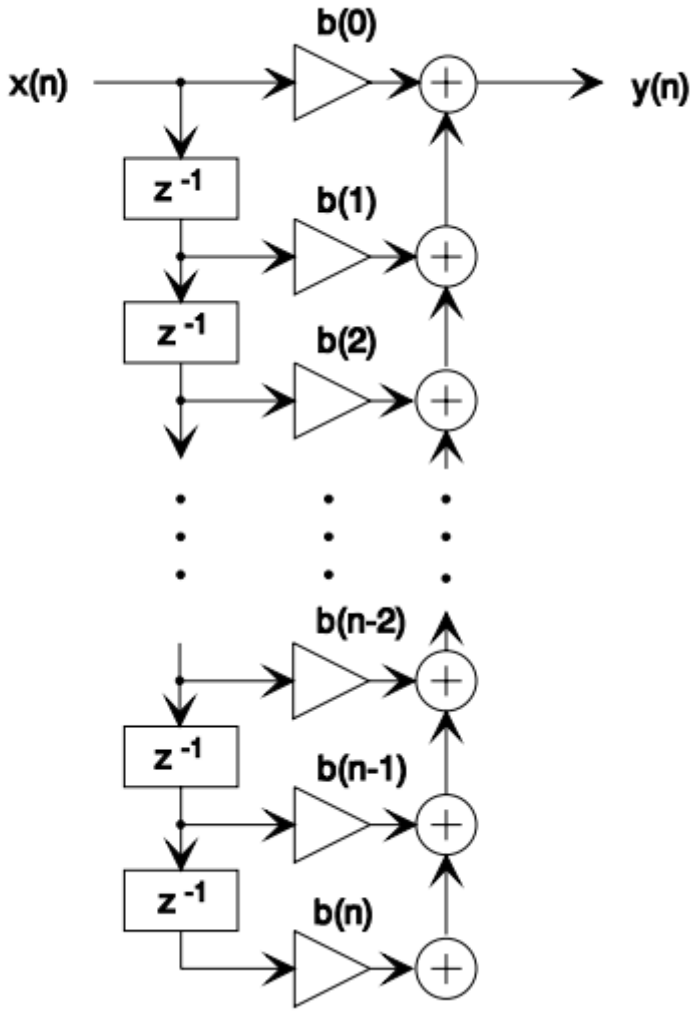


KUVA 7. Kaistanpäästösuodatin (6)

Seuraavissa luvuissa tutustutaan edellä mainittujen suodattimien toteutukseen digitaalisesti FIR(finite response)- ja IIR(infinite response)suodattimilla.

FIR-suodatin

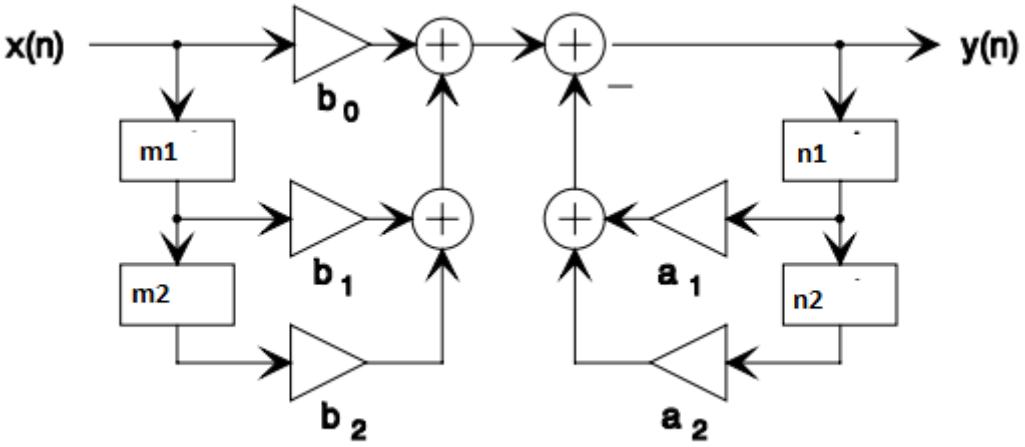
FIR-suodatin eli äärellisen impulssivasteen suodatin on suodatin, jossa ei ole takaisinkytkentää (7) (kuva 8). Tällöin suodattimen tulo ei riipu signaalin aikaisemmista arvoista. FIR-suodattimen toimintaperiaatetta on havainnollistettu liitteessä 1.



KUVA 8. FIR-suodatin (7)

IIR-suodatin

IIR-suodatin eli äärettömän impulssivasteen suodatin on suodatin, jossa on takaisinkytkentä(7)(kuva 9). Tällöin suodattimen tulo riippuu signaalin aikaisemmista arvoista. IIR-suodattimen toimintaperiaatetta on havainnollistettu liitteessä 2.



KUVA 9. IIR-suodatin (7)

2.3 Huomioitavia asioita äänen toistamisessa

Kotiteatteri- ja autohifijärjestelmissä käytetään useita erilaisia kaiuttimia, joista jokainen toimii parhaiten tietyillä äänentaajuuksilla. Tämän vuoksi on tärkeää poistaa ei-halutut äänentaajuudet oikeanlaisilla suodattimilla.

2.3.1 Kaiuttimet

Kaiutintyypit jaotellaan yleisesti neljään eri kategoriaan: diskantteihin, keskiääniin, basso/midbassoihin sekä subwoofereihin. Näistä jokainen vaatii omanlaisensa suodattimen, jotta niiden toiminta olisi optimaalinen. Taulukossa 7 on esitelty eri kaiutintyyppien vaatimukset yli- ja alipäästösuodatukselle.(8).

TAULUKKO 8. Elementtien tyypilliset jakotaajuudet

Elementti	Ylipäästötaajuus	Alipäästötaajuus
Subwoofer	–	Alle 100 Hz
Basso, midbasso	50–100 Hz	200–800 Hz
Keskiääni	300–800 Hz	3000–8000 Hz
Diskantti	3000–8000 Hz	–

Taulukosta näkyy, että subwoofer vaatii alipäästösuodatuksen, diskantti ylipäästösuodatuksen ja basso/midbasso sekä keskiäänet vaativat kaistanpäästösuodatuksen.

2.3.2 Kaiuttimien asettelu ja niille tulevan äänen suodatus

Kaiuttimien asettelulla on tärkeä merkitys tilaäänen saavuttamiseksi. Yleisimmissä kotiteatterijärjestelmissä on käytössä 6 kaiutinta, jotka on sijoitettu eri puolille huonetta. (Kuva 10).

KUVA 10. Kaiuttimien sijoittelu(9).

Kuvan 9 vasen etukaiutin (Front Left Speaker) ja oikea etukaiutin (Front Right Speaker) on suunniteltu toistamaan koko taajuusaluetta lukuunottamatta matalimpia bassotaajuuksia, joiden toistamisen hoitaa Subwoofer. Näille kauttimille riittää ylipäästösuodatin poistamaan alle 100 Hz:n taajuudet. Subwooferin tarkoitus on toistaa 20-120 Hz:n taajuuksia, joita olisi muuten vaikea toistaa järjestelmän muilla kaiuttimilla (10). Subwooferille tarvitaan siis alipäästösuodatin. Keskikaiuttimen (Center Speaker) tarkoitus on toistaa puhetta(11). Tämän takia kaiuttimelle tulevasta äänestä suodatetaan kaistanpäästösuodattimella alle 500 Hz:n ja yli 5000 Hz:n taajuudet(11). Vasen takakaiutin (Rear Left Speaker) ja oikea takakaiutin (Rear Right Speaker) on suunniteltu toistamaan 100 Hz-7000 Hz:n taajuuksia(1). Tämän takia kaiuttimelle tulevasta äänestä suodatetaan kaistanpäästösuodattimella alle 100 Hz:n ja yli 7000 Hz:n taajuudet Äänen toistaminen tehdään myös 15-30 ms:n viiveellä, jotta kuulija tulkitsee äänen heijastuvat huoneen seinistä(1). Tämä vahvistaa tilaäänen saavuttamista.

2.3.3 Yhteenveto

Oli kyseessä sitten autohifi- tai kotiteatterijärjestelmä samat yleiset huomioitavat asiat pätevät kaikille audiojärjestelmille. Todenmukaisen äänen toistaminen vaatii aina useita eri taajuuksille suunniteltuja kaiuttimia. Nämä kaiuttimet pystyvät toistamaan vain niille suunniteltuja taajuusalueita, joten oikeanlainen suodatus on tarpeellinen. Lopuksi tilaäänen saavuttamiseksi kaiuttimien sijoittelu ja viiveiden säätö on tärkeää. Kaikki nämä voidaan nykyään hoitaa hyvin digitaalisella signaalinkäsittelyllä.

3 OHJELMALLINEN TOTEUTUS

FIR- ja IIR-suodattimien toteutus ohjelmallisesti tapahtuu ensin suunnittelemalla suodatin jollain suodattimen suunnitteluohjelmalla kuten MatLabin Filter Design Toolilla. Suunnitteluohjelmassa voidaan määritellä suodattimen tyyppi, suodatettavat taajuudet ja päästö- sekä estokaistan vaimennuskertoimet. Huomioitavaa on myös, että suodattimen näytteistystaajuus tulee olla sama millä suodatettava data on tallennettu. Suodattimen suunnitteluohjelma antaa tällöin suodattimen kertoimien määrän ja niiden arvot.

Yleisesti voidaan sanoa, että IIR-suodattimen toteutus ohjelmallisesti on helpompaa kuin FIR-suodattimen toteutus, sillä samanlainen IIR-suodatin sisältää vähemmän kertoimia kuin samanlainen FIR-suodatin. Tämän johtuu sen sisältämästä takaisinkytkennästä. FIR-suodattimen käyttöä voidaan suositella jos suodattimen suunnittelun jälkeen huomataan, että kertoimien määrä on hyvin pieni.

Kun kertoimet suotimelle on saatu, pitää ne siirtää ohjelmallisesti luettavaan muotoon. Liitteissä 1 ja 2 on käyty läpi FIR- ja IIR-suotimien toimintaperiaatteet. Niiden perusteella suotimien toiminta on vain kerto-, yhteen- ja vähennyslaskuja. Ohjelmallisesti tämä on helppo toteuttaa. Ensiksi suotimen muistipaikoiksi luodaan float-muuttujia(Kuva 11).

```
//Suodatukseen liittyviä muttujia
float output1, output2;
float m1, m2, m3, m4 = 0;
float n1, n2, n3, n4 = 0;
```

KUVA 11. Ohjelmallisesti toteutetun IIR-suodattimen muistipaikat muuttujina.

Seuraavaksi toteutetaan laskutoimitukset, mitkä sisältävät suodattimesta saadut kertoimet sekä luodut muistipaikat. Laskutoimitusten jälkeen muistipaikoissa olevia arvoja siirretään kuten liitteissä 1 ja 2.(Kuva12).

```
void suodatus()
{

  output1 = (1*nopeus + -2*m1 + 1*m2 - -1.9400689500382613*m3 - 0.94337193595385138*m4)*0.97086022149802809;
  m4 = m3;
  m3 = output1;
  m2 = m1;
  m1 = nopeus;

  output2 = (1*output1 + -1*n1 + 0*n2 - -0.94332513688519215*n3 - 0*n4)*0.97166256844259613;
  n2 = n1;
  n1 = output1;
  n4 = n3;
  n3 = output2;



}
```

KUVA 12. IIR-suodattimen ohjelmalliset laskutoimitukset.

Laskutoimituksissa esiintyvä nopeus-muuttuja on suodatettava signaali. Huomattavaa on myös, että kuvan 11 suodatin on kaksiosainen. Tämän vuoksi suodatettava signaali pitää ajaa laskutoimitusten läpi kahteen kertaan. Tämän vuoksi se sisältää output1- ja output2-muuttujat. Lopulta output2 sisältää suodatetun signaalin.

4 MP3-TIEDOSTOMUOTO

4.1 Historia

MP3-tiedostomuoto on standardi, joka pakkaa minkä tahansa äänitiedoston pienempään tilaan. Se on osa MPEG(Motion Pictures Expert Group) -perhettä, joka sisältää useita eri standardeja äänen ja videon toistamiseen.(12).

Saksalainen Fraunhofer Institute aloitti projektin nimeltä EUREKA project EU147, Digital Audio Broadcasting (DAB), jonka tarkoituksena oli tutkia ja kehittää korkealaatuinen ja matalalla bittinipeudella toimiva audiokooderi. Projektiin osallistuivat henkilöt nimeltä Bernhard Grill, Karl-Heinz Brandenburg, Thomas Sporer, Bernd Kurten, ja Ernst Eberlein, joiden nimissä on nykyinen MP3 patentti.(12).

4.2 Rakenne

CD-laatuinen ääni on tyypillisesti tallennettu 16-bittisillä näytteillä ja 44,1 kHz:n näytteistystaajuudella, joka on noin kaksi kertaa suurempi kuin suurin taajuus minkä ihminen pystyy kuulemaan. Tämän vuoksi yhden sekunnin mittainen CD-laatuinen äänitiedosto olisi kooltaan 1,4 Mbit. MP3-tiedostomuoto mahdollistaa tiedoston pienentämisen jopa yhteen 12 osaan alkuperäisestä.(12).

MP3-tiedosto on jaettu useisiin pieniin kehyksiin, joista jokainen on 0,026 sekuntia pitkä. Kehysten koko voi vaihdella tiedoston bittinopeuden mukaan, mutta ajallisesti ne ovat aina yhtä suuret. Ensimmäiset neljä tavua joka kehyksessä sisältävät tietoa MP3-tiedoston ominaisuuksista kuten tiedoston bittinopeuden. Loput tavuista sisältävät tiedon itse äänestä(13).

Ensimmäiset neljä tavua on jaettu 13 osaan. Kuvassa 13 on esitetty kaikki osat kirjaimilla A-M.

KUVA 13. MP3 kehyksen neljä ensimmäistä tavua(13).

Kirjaimella A merkityt 11 bittiä ovat aina asetettu ykkösiksi. Tällä yritetään varmistaa, että MP3-kehyksen alku aina löydetään. Tämä ei kuitenkaan aina varmista kehyksen löytymistä sillä samanlaisia bittijonoja voi löytyä muuallakin kehyksessä. Ongelmaa voidaan ehkäistä tarkistamalla koko kehys, jotta varmistutaan siitä missä kohtaa kehys on alkanut, mutta tämä tapa on todella aikaa vievää.(13).

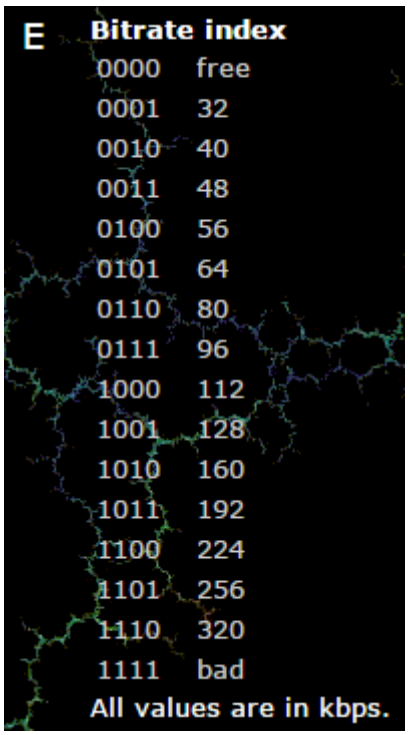
Seuraavat kaksi bittiä, jotka on merkitty kirjaimella B, kertovat MPEG versiosta. Nämä bitit ovat yleensä kaikissa MP3-tiedostoissa asetettu ykkösiksi, joka vastaa tällöin MPEG versiota 1(13).

Kirjaimella C merkityillä biteillä on ilmoitettu MP3-tiedoston Layerin taso. Useimmissa tiedostoissa nämä on asetettu arvoon 01, joka vastaa Layeriä 3. (13).

Kirjaimella D merkityt bitit ilmoittavat, onko kehyksissä käytössä CRC-tarkistusta. CRC-tarkiste on 16 bittiä pitkä, ja se sijoitetaan kehyksen ensimmäisen neljän tavun jälkeen. Yleensä CRC-tarkistetta ei käytetä. Tällöin D:n arvoksi asetetaan ykkönen.(13).

Kirjaimella E merkityt neljä bittiä kertovat MP3-tiedoston bittinopeudesta. Taulukossa 9 on esitetty kaikki mahdolliset bittinopeudet.(13).

TAULUKKO 9. MP3-tiedoston mahdolliset bittinopeudet(13).



E	Bitrate index
0000	free
0001	32
0010	40
0011	48
0100	56
0101	64
0110	80
0111	96
1000	112
1001	128
1010	160
1011	192
1100	224
1101	256
1110	320
1111	bad

All values are in kbps.

Kirjaimella F merkityt bitit kertovat MP3-tiedoston näytteistystaajuuden. Yleensä tämä on asetettu arvoon 11, joka vastaa 44100 Hz:n näytteistystaajuutta.(13).

Kirjaimella G merkitty bitti kertoo sisältääkö MP3-tiedosto tiedon täyttämistä(padding). Esimerkiksi normaalit 417 tavun kokoiset kehykset eivät anna tasan 128kbps bittinopeutta, mutta lisäämällä ylimääräisen tavun kehyksen loppuun antaa juuri oikean bittinopeuden. Täyttämisellä ei ole usein väliä, joten bitin G voidaan asettaa joko arvoon nolla tai yksi.(13).

Kirjaimella H merkitty bitti kutsutaan nimellä Private Bit. Tämä voidaan vapaasti asettaa mihin arvoon tahansa, koska sillä ei ole vaikutusta MP3-tiedostoon. Private Bit:iä voidaan esimerkiksi käyttää joidenkin applikaatioiden yhteydessä tekemään jokin haluttu toiminto.(13).

Seuraavat kaksi bittiä, jotka ovat merkitty kirjaimella I, kertovat onko ääni stereona vai monona. Arvo 11 kertoo tiedoston toistavan äänen monona. Arvot 00, 01 ja 10 ovat kolme erilaista tapaa toistaa ääni stereona. Yleisin näistä on 01 eli Joint Stereo.(13).

Kirjaimella K merkitty bitti kertoo onko tiedosto suojattu tekijänoikeudella. Bitti on arvossa 1, jos tiedosto on suojattu.(13).

Kirjaimella L merkitty bitti kertoo onko tiedosto kopioitu vai alkuperäinen. Bitti on arvossa 1, jos tiedosto on alkuperäinen.(13).

Viimeiset kaksi bittiä, jotka on merkitty kirjaimella M, kertovat onko tiedostolla painotusta taajuuksille, jotka ylittävät 3200 Hz(13).

Kehysten lisäksi MP3-tiedoston lopussa tai alussa on TAG niminen osa, johon voidaan tallentaa tietoja tiedostosta esimerkiksi kappaleen nimi, artisti, albumi ja genre.

5 POHDINTA

Opinnäytetyön ensimmäisessä osassa oli tarkoitus tehdä selvitystyötä omalle opinnäytetyön aiheelle. Selvitystyön aikana oli tarkoitus tutustua digitaaliseen signaalinkäsittelyyn ja äänentoistamiseen.

Työssä selvitettiin digitaalisen signaalinkäsittelyn eri muotoja. Äänen toistamisen kannalta merkittäviä asioita käytiin läpi kuten suodattimien käyttö ja kaiuttimien asettelu. Työssä tutustuttiin myös MP3-tiedostomuodon rakenteeseen, joka on yleisin tiedostomuoto äänen tallentamiseen.

Aihe oli mielenkiintoinen ja tuttu omien harrastusten kautta, mutta työ jäi haasteellisuudeltaan aika matalaksi.

LÄHTEET

1. Smith, Steven W. 2001. The Scientist and Engineer's Guide to Digital Signal Processing. Chapter 1. Newnes.

2. Lernet, Sandy 1999. The Dilettante’s Dictionary. Saatavissa: http://www.dilettantesdictionary.org/index.php?search=1&searchtxt=analog-to-digital . Hakupäivä 30.3.2015

3. Smith, Steven W. 2001. The Scientist and Engineer's Guide to Digital Signal Processing. Chapter 3. Newnes.

4. DSP Design Made Easy. Machine Design. 26.6.1990. ABI/INFORM Complete.

5. Bartlett, Bruce. Digital Recording Does Not Chop Up Your Music. Saatavissa: http://archive.l2pnet.com/blogs/digital-recording-does-not-chop-your-music-11102010 Hakupäivä 31.3.2015

6. Sievers, Beau. A Young Person's Guide to the Principles of Music Synthesis. Saatavissa: http://beausievers.com/synth/synthbasics/ Hakupäivä 7.4.2015

7. Jokinen, Raimo 2003. Digitaalinen Signaalinkäsittely Lineaariset Järjestelmät. Turku Institute of Technology. Saatavissa: http://www.hovirinta.fi/audio/suunnittelu/teoriaa/DSP/Digitaalinen%20signaalink%E4sittely%2003.pdf Hakupäivä 7.4.2015

8. Jakosuodattimen toiminta. AutoSound Technical Magazine. Saatavissa: http://www.autosound.fi/index.php/jakosuotimet.html Hakupäivä 10.4.2015

9. Wilson, Tracy ja Harris, Tom 2001. How Home Theater Works. Saatavissa: http://electronics.howstuffworks.com/home-theater1.htm Hakupäivä 10.4.2015

10. Mikkola, Jari 2005. Hifiopas. Saatavissa: http://www.students.tut.fi/~jmikkola/hifiopas/subwoofer.html#yleista Hakupäivä 12.4.2015

11. 2011. The Center speaker is the most important speaker in your system for Home Theater. Roberts Audio Video. Saatavissa: http://robert-sav.com/blog/2011/06/the-center-speaker-is-the-most-important-speaker-in-your-system-for-home-theater/ Hakupäivä 12.4.2015

12. Bellis, Mary. The History of MP3. Saatavissa: http://inventors.about.com/od/mstartinventions/a/MPThree.htm Hakupäivä 24.4.2015

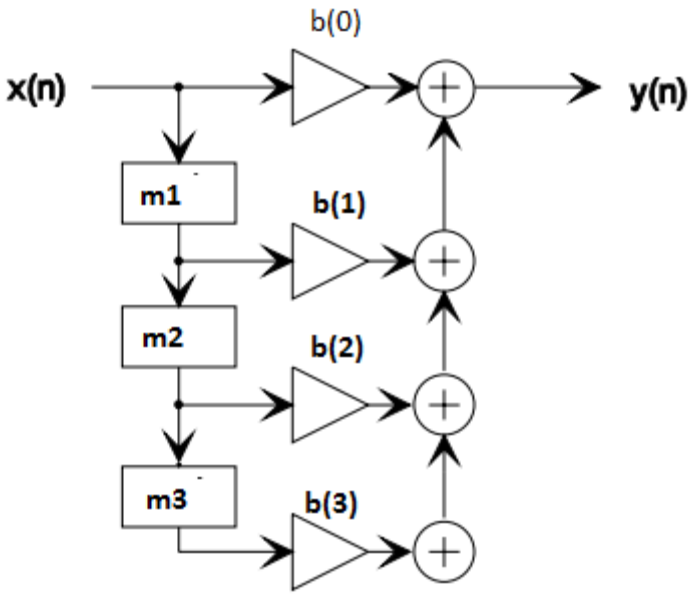
13. Inside MP3. Saatavissa: http://www.multiweb.cz/twoinches/mp3inside.htm Hakupäivä 24.4.2015

OULU UNIVERSITY OF APPLIED SCIENCES

LIITE 1. FIR-SUODATIN

FIR-suodatin koostuu muistipaikoista ja kertoimista. Signaalin, jota halutaan suodattaa, syötetään suodattimeen näyte kerrallaan. Näyte kerrotaan suodattimen arvolla b(0), jonka jälkeen muistipaikoissa olevat arvot kerrotaan niitä vastaavilla kertoimilla b(1), b(2) jne. Nämä arvot lisätään yhteen ja lopuksi muistipaikoissa olevia arvoja siirretään yhden eteenpäin.

Prosessia on havainnollistettu kuvien kanssa alla.



Kuva 7. FIR-suodatin.

Yllä olevassa FIR-suodattimessa on neljä kerrointa(b(0) – b(3) ja kolme muistipaikkaa(m1 – m3). Kertoimien ja muistipaikkojen arvot on esitetty taulukossa 1.

Taulukko 1. Suotimen kertoimet ja muistipaikat.

Suotimen kertoimet	**Muistipaikkojen arvot**

OULU UNIVERSITY OF APPLIED SCIENCES

b(0)	1	m1	0
b(1)	0,4	m2	0
b(2)	0,6	m3	0
b(3)	0,2		

Suotimeen syötetään seuraavanlaisen signaalin ensimmäinen näyte(taulukko 2).

Taulukko 2. Signaalin näytteet..

Signaali	
Näyte	Arvo
1	12
2	11
3	9
4	6
5	5
6	6
7	10
8	13
9	15
10	21
11	22
12	22
13	22
14	18
15	17
16	16
17	11
18	8
19	4
20	1

Näyte 1 kerrotaan kertoimella b(0). Muistipaikan 1 arvo kerrotaan kertoimella b(1). Muistipaikan 2 arvo kerrotaan kertoimella b(2). Muistipaikan 3 arvo kerrotaan kertoimella b(3). Lopuksi nämä arvot summataan yhteen.

Lähdön arvoksi saadaan:

12 * 1 + 0 * 0.4 + 0 * 0.6 + 0 * 0.2 = 12

Lähdön arvo otetaan talteen omaan taulukkoonsa. Tämän jälkeen näytteen 1 arvo siirretään muistipaikkaan 1. Muistipaikan 1 arvo siirretään muistipaikkaan 2. Muistipaikan 2 arvo siirretään muistipaikkaan 3.(Taulukko 3).

Taulukko 3. Muistipaikkojen arvot.

Muistipaikkojen arvot	
m1	12
m2	0
m3	0

Suodatin on nyt valmis ottamaan vastaan seuraavan näytteen. Lähdön arvo saadaan samoilla perusteilla kuin aikaisemminkin:

11 * 1 + 12 * 0.4 + 0 * 0.6 + 0 * 0.2 = 15.8

Muistipaikkoja siirretään taas ja laskutoimituksia toteutetaan niin kauan kuin näytteitä vain on(taulukko 4).

Taulukko 4. Muistipaikkojen arvot.

Muistipaikkojen arvot	
m1	11
m2	12
m3	0

19 * 1 + 11 * 0.4 + 12 * 0.6 + 0 * 0.2 = 30.6

.

.

.

Lopullinen suodattimesta saatu signaali on esitetty alla taulukossa 5.

Taulukko 5. Suodatettu signaali.

Lähtö	
Näyte	Arvo
1	12
2	15,8
3	30,8
4	18,6
5	15
6	13,4
7	16,6
8	21,6
9	27,4
10	36,8

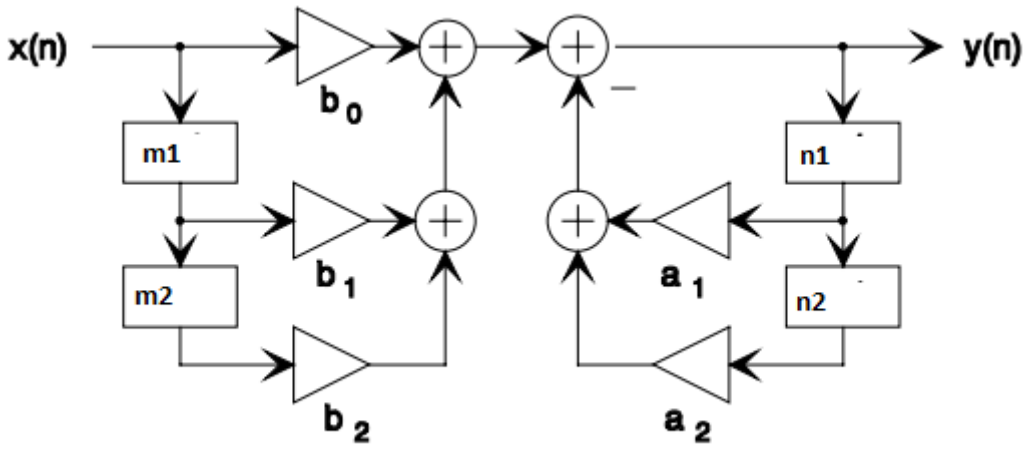
Yllä käytetty signaali sekä suodattimen kertoimet ovat keksittyjä eivätkä todennäköisesti kuvaa mitään käytännöllistä signaalia tai suodatinta. Suodattimen toiminta ja laskutoimitukset ovat silti samat.

OULU UNIVERSITY OF APPLIED SCIENCES

LIITE 2. IIR-SUODATIN

IIR-suodatin koostuu muistipaikoista ja kertoimista. Signaalin, jota halutaan suodattaa, syötetään suodattimeen näyte kerrallaan. Näyte kerrotaan suodattimen arvolla b0, jonka jälkeen muistipaikoissa olevat arvot kerrotaan niitä vastaavilla kertoimilla b1 ja b2. Nämä arvot lisätään yhteen. Tämän jälkeen saadusta arvosta vähennetään takaisinkytkennästä saadut luvut. Nämä luvut saadaan kertomalla muistipaikkojen n1 ja n2 luvut niitä vastaavilla a-kertoimilla. Lopuksi muistipaikoissa olevia lukuja liikutetaan, kuten FIR-suodattimessa.

Prosessia on havainnollistettu kuvien kanssa alla.



Kuva 8. IIR-suodatin (7)

IIR-suodatin on toiminnallisesti samanlainen, kuin FIR-suodatin sillä erotuksella, että takaisinkytkennän takia matemaattinen laskutoimitus sisältää erouksen. Taulukossa 6 on esitetty suodattimen kertoimet ja muistipaikkojen arvot. Käytettävän signaalin näytteet saadaan taulukosta 2.

OULU UNIVERSITY OF APPLIED SCIENCES

Taulukko 6. IIR-suodattimen kertoimet ja muistipaikkojen arvot.

Suotimen kertoimet		**Muistipaikkojen arvot**	
b0	1	m1	10
b1	0,4	m2	13
b2	0,6	n1	0
a1	0,2	n2	0
a2	0,3		

Tällöin lähdön arvoksi saadaan

12 * 1 + 10 * 0.4 + 13 * 0.6 - 0 * 0.2 – 0 * 0,3 = 23,8

Kun muistipaikkojen arvoja aletaan siirtämään, on huomattava, että muistipaikan n1 arvoksi ei tule muistipaikan m2 arvo, vaan juuri laskettu lähdön arvo. Tämän vuoksi IIR-suodatinta kutsutaan äärettömän impulssivasteen suodattimeksi, koska suodattimen aikaisemmat arvot jäävät vaikuttamaan sen käyttäytymiseen. Muistipaikkojen uudet arvot on esitetty taulukossa 7.

Taulukko 7. Muistipaikkojen arvot.

Muistipaikkojen arvot	
m1	12
m2	10
n1	23,8
n2	0

OULU UNIVERSITY OF APPLIED SCIENCES

Markus Varila

IMPROVING SOFTWARE TESTING AUTOMATION AND PREPARING FOR CLOUDIFICATION

Markus Varila

IMPROVING SOFTWARE TESTING AUTOMATION AND PREPARING FOR CLOUDIFICATION

Markus Varila
Thesis
Spring 2017
Degree Programme in Information Technology
Oulu University of Applied Science

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Wireless Devices

Author: Markus Varila
Title of the bachelor's thesis: Improving Software Testing Automation and Preparing for Cloudification
Supervisors: Kari Jyrkkä, Harri Maatela, Lazo Eric
Term and year of completion: Spring 2017 Number of pages: 31 + 1 appendices

This Bachelor's thesis was commissioned by Nokia Oyj to streamline automated testing in a software testing team. The main objective was to identify problems which cause unnecessary manual work in everyday work and implement solutions for them.

As a result of this study, identified problems were solved by implementing Python scripts which would handle the manual tasks automatically. It was also planned to use these scripts in a cloud test environment which is currently in development.

Keywords: software testing, test automation, base station, Python

PREFACE

The idea for this thesis work came from our team manager Lazo Eric from Nokia Oyj. I have been working in the team for quite some time already doing project work and as a summer trainee. This thesis was a natural continuation of the work I have been doing for the past year.

I would like to thank Kari Jyrkkä, the tutoring teacher, for all the support I have received during this thesis.

Oulu, 26.2.2017

Markus Varila

CONTENTS

GLOSSARY

5G	5th generation mobile networks
CRT	Continuous Regression Testing
CIT	Continuous Integration Testing
ISO	International Organization for Standardization
SDLC	Software Development Life Cycle
STLC	Software Testing Life Cycle
SVN	Subversion
TDD	Test Driven Development
VCS	Version Control System

1 INTRODUCTION

The aim of this thesis was to become acquainted with software testing and especially with testing automation to identify problems with the current implementation of automated testing in a testing team and offer as many solutions to these problems as possible. This thesis was commissioned by Nokia Oyj which develops base stations. Software testing focuses on these products.

The growing need for automated regression testing is essential to a complex software system. The team in Nokia Oyj had tools for automated software testing but there still existed many manual steps in everyday work that made fully automated testing not viable.

Nokia Oyj is a global leader in technologies that connect people and things (1). Nokia offers base station solutions that work with all current generation cellular technologies. Their base station solutions are also 5G-ready, offer Internet-of-Things solutions and unlimited scalability with high energy efficiency. (2).

2 SOFTWARE TESTING

This chapter focuses on explaining what is software testing, why it is needed and what different kind of testing exist.

2.1 Purpose of software testing

Software testing is needed because humans make mistakes. These mistakes can be costly or even dangerous and by assuming that mistakes are made, people should be prepared for them. (3).

It is not usually efficient to check one's own work for mistakes because the same mistakes can be made while checking it as was done during making it. Thus, it is useful for someone else to check the work. (3).

Comprehensive testing will ensure that the product maintains a high quality, is reliable and has no or very few defects. High quality and reliability will ensure customer satisfaction and having no defects in the software is beneficial because defects can be very expensive to correct at the later stages of development. (3).

2.1.1 Defects

Defects can raise in different parts of the development (4). Having a good understanding of the design process is a key element for understanding when and how defects may raise.

The development starts with a requirement which is made from the customer's needs. This is followed by design that meets the requirements. Finally, a software is made to fit the design. (4).

If mistakes have been made during the coding of the product, a defect is raised and the final product will have bugs in it. These are usually easiest to spot during testing and they can be fixed before delivery. Mistakes can also happen during the design phase of development. In this case, the design does not meet the requirements. These defects are harder to spot because the developer might

make a bug free software that functions as the design states but it does not meet the requirements given by the customer. (4).

For a tester, it is also important to look at the functionality of the software from the customer's point of view. Even if the functionality meets the design, the functionality might be odd, confusing or unintuitive to the customer. In these cases, the tester should be in contact with the designer to see if the design can be changed. This is a way to correct defects that may happen at the design phase of development.

Finally, defects may also raise if the requirements do not fit the customer's needs. In this case, all testing may pass but the developed product is not what the customer wanted. (4).

2.1.2 Testing independence

There are levels of independence depending on who is testing the software. This independence can avoid bias towards the product which leads to a more efficient way of finding defects. The amount of independence can usually be divided into four levels. (5).

1. Tests are executed by the person who wrote the code
2. Tests are executed by a person in the same team
3. Tests are executed by another team, for example an independent test team
4. Tests are executed by another organization or company

The first level on the list has the lowest amount of independence and the last level on the list has the highest one.

Benefits of high level of independence means that the tester can find more or different kinds of defects that would not be likely at the lower levels. A tester at a higher level of independency can also report the findings more honestly and without the concern for reprisal. (5).

High independence also has risks. Testers and developers might become isolated entities without much communication between each other. Pointing out

problems may feel hostile and in worst cases, a tester may not report a problem because of this. Independent test teams may also seem like bottlenecks or sources of delay in the development process. (5).

2.1.3 Psychology of testing

The mindsets of a developer and a tester are different. During the development a process, developer works to solve problems and to make a product that meets the given requirements or specifications. A tester, however, is looking for faults in the software. (6).

Humans are prone to getting annoyed, upset or angry when they are pointed out the mistakes they have made. A tester needs to keep this in mind when reporting the found defects in the product because the people responsible might react defensively to the report. Therefore, reporting should be as objective and polite as possible. (6).

2.2 Testing process

Software developers follow the Software Development Life Cycle (SDLC). Testers follow a similar process called the Software Testing Life Cycle (STLC). This process describes actions taken by the test team from the beginning of the project till the end of it. A process is visualized in figure 1. (7).

STLC consists of 6 steps:

1. Requirement analysis
2. Test Planning
3. Test case development
4. Environment Setup
5. Test Execution
6. Test Cycle Closure

Each steps has its own Entry Criteria as well as Exit Criteria. These describe the minimum set of conditions that must be met in order to start and end the step. (7).

During the Requirement analysis, System Requirements are analyzed to determine what kinds of testing techniques and testing types are needed. The most important features that need to be prioritized are identified and automation feasibility is analyzed. Details about needed test environments are also identified. (7).

During the Test Planning, the testing effort, resources and assigning responsibilities are decided. The Testing approach, plan, strategy and tools are also decided. (7).

Test Case Development consists of writing the test cases and possible automation scripts. If automation scripts are developed, they also need to be verified. (7).

The Test Environment setup includes setting up and installing all the required hardware and software to test the application. Finally, a smoke test is performed to the test environment. Smoke testing is described in chapter 2.4.4. The test environment setup can usually be done alongside the Test Case Development step. (7).

During the Test Execution step, planned test cases are executed in the test environment and found defects are reported to the developers. The Developers will fix the defect and retesting is done to verify the correction. (7).

Finally in the Test Cycle Closure step, the test team discusses the testing process and tries to find parts that could be improved. This ensures that future testing will be more efficient. (7).

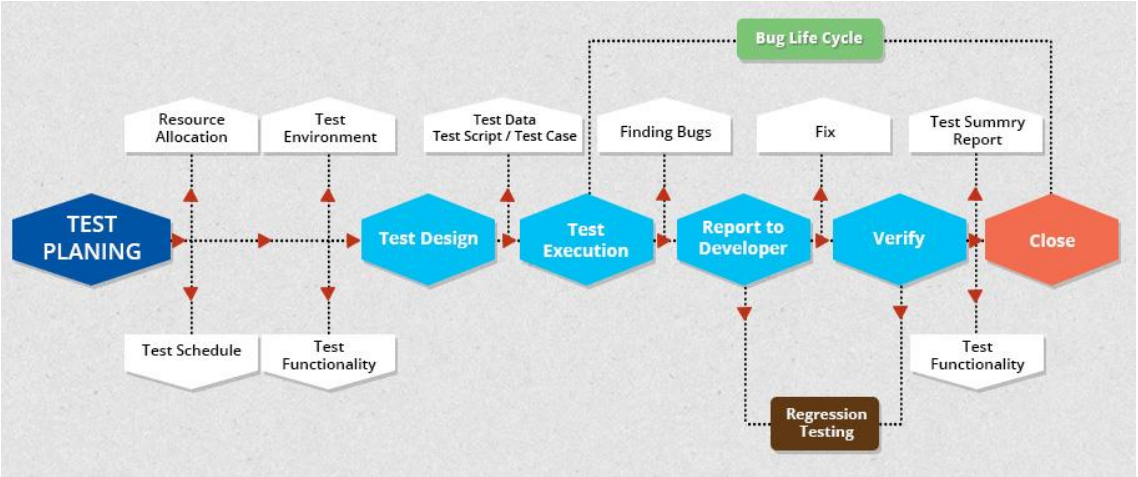


FIGURE 1. Testing process (8).

2.3 Benefits of software testing

This chapter focuses on describing the benefits of software testing and the tools and methods used to perform testing.

2.3.1 Principles of testing

There exists seven principles that software testing should follow.

1. Testing shows the presence of bugs

Testing can reveal the presence of one or more defects in the system but it cannot prove that the system is error free. Therefore, test cases should be designed in a way that they find as many defects as possible. (9, 10)

2. Exhaustive testing is impossible

Testing all the possible scenarios for a system is impossible. Instead, tests should be designed based on the risk and priorities. (9, 10)

3. Early testing

Testing should be started as early as possible because correcting faults is always cheaper the earlier in the development process they are found. (9, 10)

4. Defect clustering

Based on the Pareto Principle (11), defect clustering means that small set of modules in a system usually contain the most defects. (9, 10)

5. The pesticide paradox

This principle dictates that when a certain test is run over and over again, it will not find any more new defects. Old tests should be changed to reflect the new functionality of ever changing systems and/or new tests should be written to overcome the pesticide paradox. (9, 10)

6. Testing is context dependent

The type of testing, techniques and methodologies change depending on the application being tested. Some applications need to be tested more because they might have safety concerns otherwise e.g. popular applications might need a great load testing. (9, 10)

7. Absence of errors fallacy

The absence of errors is not an indication that software is error free (9, 10).

2.3.2 Finding defects

One aspect of the quality of the software is the absence or reasonable amount of defects. The customer expects the product to work reliably and error free. Therefore, finding defects before the product is delivered is essential for the customer satisfaction. (12).

The cost of correcting defects is determined by the phase where it is found. The earlier in the development process the defect is found, the easier it is to correct. If a defect is found in live use, the cost of correction is extremely high as seen in figure 2. This is because changes may need to be made to the requirements, design and implementation which is followed by testing to make sure that the quality of the product is still good. Depending on the severity of the defect found

during live use, it is common not to implement a correction because the cost would simply be too expensive. (13).

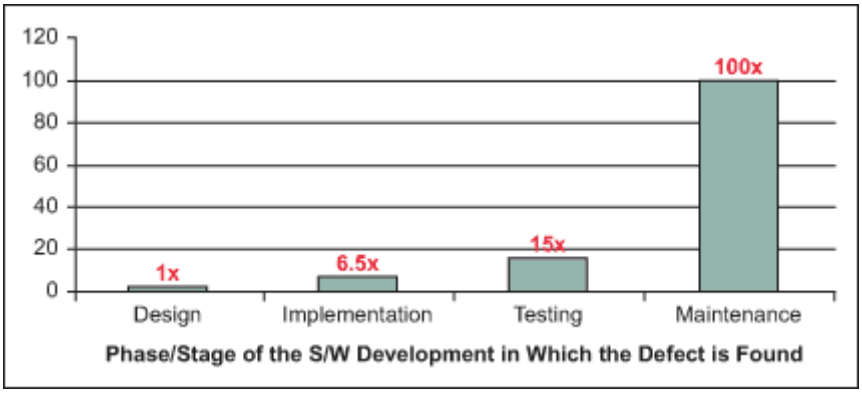


FIGURE 2. Relative cost to fix software defects (14).

2.3.3 Software quality

The ISO 8402 1994 standard defines the quality as “the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs” (15)..

For software, this means that it can meet the customer’s requirements and be reliable and consistent. Software nowadays also requires after sales service in the form of maintenance. This can include bug fixes or changes in functionality. (12).

2.4 Different ways of software testing

There exists multiple different levels of software testing that test the software in different parts of the development cycle. The scope and size of the software is also vastly different depending the level of testing. (16).

2.4.1 Test Driven Development and unit tests

The Test Driven Development (TDD) is a way of programming that focuses on writing tests before writing any functional code. It is part of the agile software

developing methods and it is strictly done by the developer themselves. (17, 18).

In TDD, the developer will write a new test, which will fail. After that, they will start to write the functional code until the test passes. The code is refactored to meet the coding standards afterwards. A new test will be written and the pattern continues, as seen in the figure 3, until all the wanted functionality is present in the code and all tests pass. These tests are called unit tests. (17, 18).

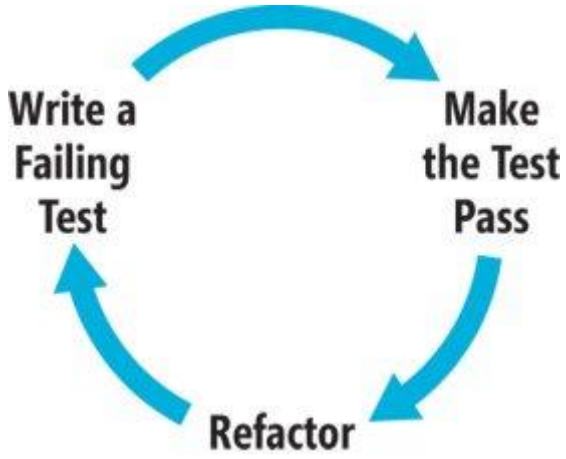


FIGURE 3. TDD process cycle (31).

TDD has many benefits. It forces the developer to think their code in small parts instead of working with a big whole. Unit tests also work as a regression test of sort. If any of the unit tests that passed before fail, the recent changes broke the code. Regression testing is explained in more detail in chapter 2.4.3. Unit tests can also work as a documentation. A developer, who comes to work on a source code that they haven't seen before, can look at the unit tests and see how the code is supposed to work instead of going to read the documentation and trying to figure out how the functionality was achieved. However, this should not be seen as a replacement for documentation but rather an addition. (17).

2.4.2 Integration testing

Integration testing refers to testing a whole system which consists of multiple components, interfaces between components or interactions between systems or hardware. Even when individual systems are tested alone, it needs to be made sure that they also work when integrated together. (19).

There are multiple integration testing techniques. In Big Bang integration testing (figure 4), everything is tested at once. All the components or modules are integrated at the same time when testing is executed. (19).

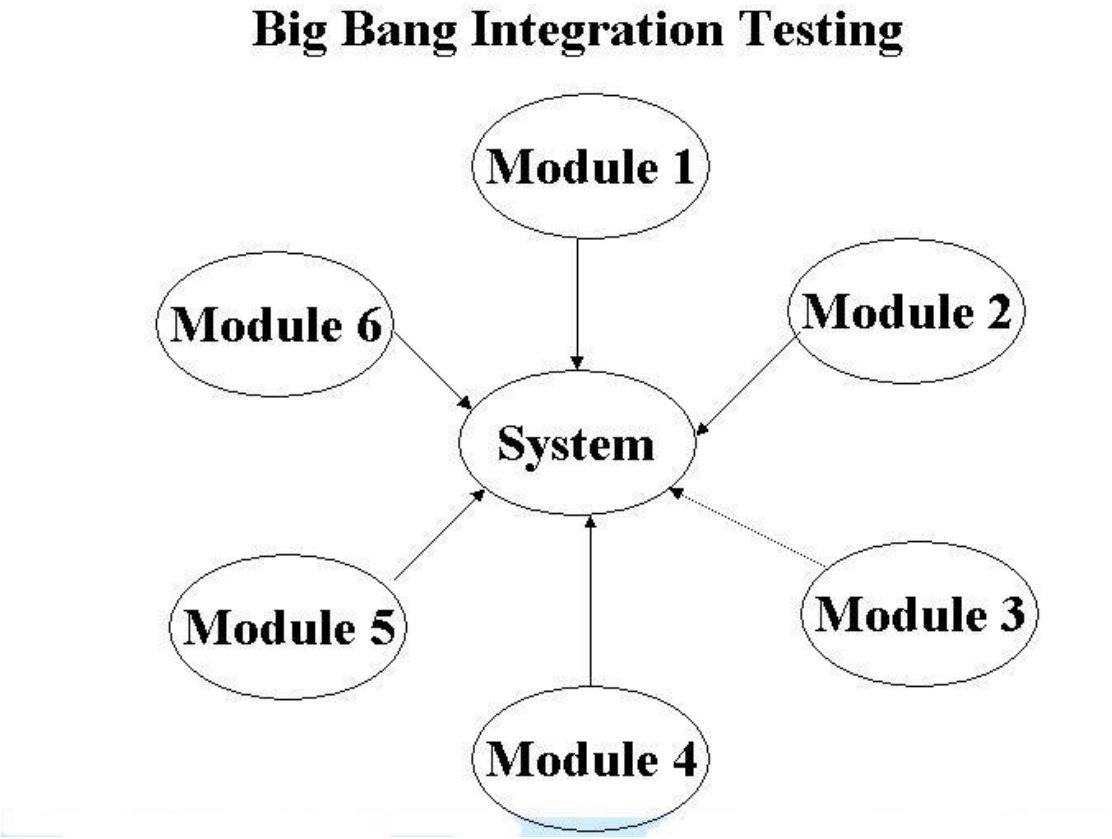


FIGURE 4. Big Bang Integration Testing (19).

An advantage of Big Bang integration testing is that all modules are already complete when the testing starts but finding the root cause of defects is harder (19).

Two other kinds of integration testing are called Top-down and Bottom-up integration testing. In the Top-down testing, the whole system is tested when a new feature or a new part of the system is introduced. It is verified that the system still works as intended and that the new part interacts correctly with the old one. Bottom-up approach focuses on the integratable part of the system and it tests if the part behaves correctly against the surrounding system. This usually involves making drivers for the surrounding parts of the system. (20).

2.4.3 Regression testing and automation

Making even the smallest changes can impact the functionality in other parts of the system. Regression testing ensures that all the existing functionality is still intact after every change in the code. Because regression testing should be done after every change, automating these tests is encouraged. Manual regression testing is time consuming, tedious and prone to errors. (21).

Automated regression tests also have disadvantages. Firstly, developing these tests takes time and effort and they are often fragile. Slight, even acceptable, changes in the system might get reported as failures. This can be fixed by maintaining the automated test scripts but this again takes time and effort. Automated regression testing will also not detect all defects. It will only detect defects it was designed to detect. Therefore it is important to decide what kinds of tests are fit for automated regression testing. (22, 23).

2.4.4 Smoke testing

Smoke testing, or Build Verification testing, is used to ensure that all the basic features of the system are working correctly. Smoke testing consists only of a very small set of tests that are executed before the build goes through more detailed testing. This ensures that if there are obvious bugs in the build, there is no time wasted on the more detailed testing. (24).

The advantages of smoke testing is that it is fast, bugs are found early and it is a form of integration testing. Smoke testing, however, is not very detailed. (24).

2.4.5 Test tools

Using different kinds of testing tools can have huge benefits in efficiency and test management. These tools can be used to manage all the running tests, to manage their results, their number of execution and other statistic. A list of different tools can be seen in figure 5. These statistic can hold valuable information about the effectiveness of the testing. Without any tools, a person interpreting these statistics might judge them wrongly or omit some results. (25).

Test execution tools can be used to assist in running the test, logging the results and reporting them. A good example of this would be automation tools which will run the wanted tests automatically. Letting automation handle the tests ensures that the tests are always run precisely and with correct data inputs every time. The probability of human error decreases dramatically. Most automated tests are also faster than the ones that a human could do. (26).

Measurement tools can also be used to measure different aspects of the system, for example, a software execution time or other performance tests. This, however, may introduce a thing called a probe effect. The probe effect means that taking the measurement will affect the measureable system, skewing the results. (26).

Using test tools can also have disadvantages. It is necessary to learn how to use the tools and how to use them efficiently. People might also become too dependent on them. Maintaining the assets required to use the tools also takes time and effort. (27).

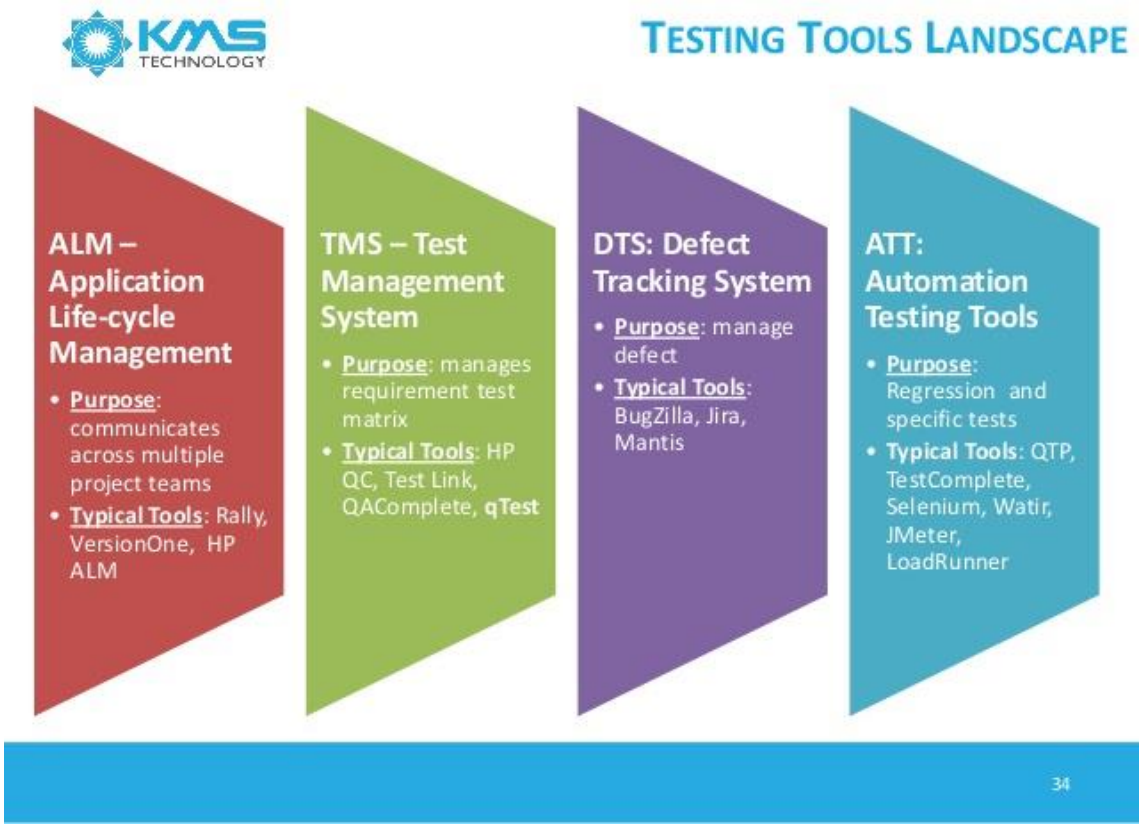


FIGURE 5. Various testing tools divided into categories (28).

2.5 Reporting defects

Once testing finds a defect it must be reported to be fixed. Reporting defects has some general guidelines that should be followed.

2.5.1 Defect report

A defect report should be clear and specific. This ensures that no time is wasted on trying to understand what is being reported. The tester should specify clearly what actions are made to produce the defect to make sure that it can be reproduced by anyone. The tester should also provide as much information as possible. Not all of this information might be used by the developer to make a fix but it will reduce the need for asking additional information from the tester. Subjective statements should be avoided from the defect report. The report should focus on facts. A good idea is to add a reference to specifications or other docu-

ments, which describe the correct behaviour of the system. Defects should always be reproduced more than once to be sure of their existence. However, this does not mean that occasional defects should not be reported. (29, 30, 31).

An example of a defect report can be found in Appendix 2.

2.5.2 Defect Life Cycle

A defect life cycle, or a bug life cycle, describes the process the defect goes through after it has been reported. This process may vary from organization to organization but the basic idea is the same. The defect will have different states during its life cycle. They are shown in figure 6. (33).

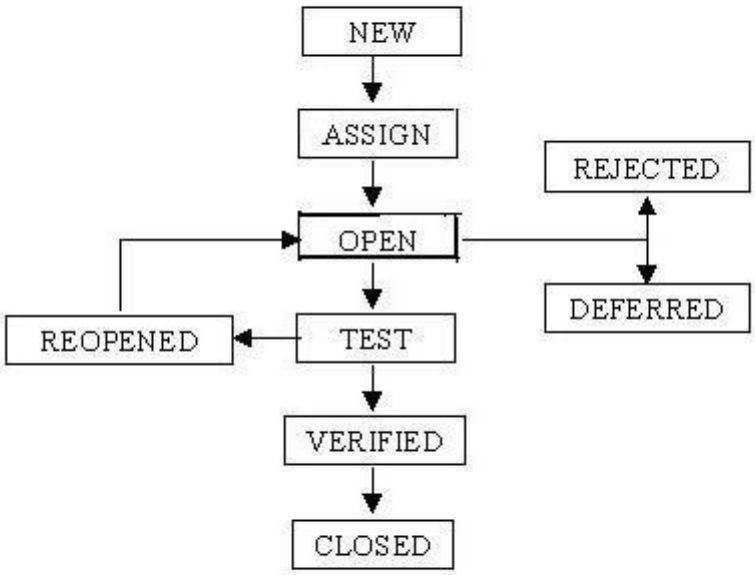


FIGURE 6. Defect Life Cycle (33).

When a defect has been reported, it will be assigned to a responsible developer or a developer team. The defect is in an “Open” state when it is being investigated and corrected. Once the changes have been made and committed, the defect will be ready for testing. The tester verifies the correction and will close the defect report if the defect is not present in the software anymore. Otherwise, they will reopen the defect report again. (33).

A defect might also be “Rejected” if the defect is not genuine. This may be because of misunderstanding the functionality by the tester or a problem with the test environment. (33).

A defect is “Deferred” if the fix for it will not be made during the current release. The decision for deferring a defect might be based on the priority and severity of the defect. There might exists more severe defects that must take priority over other defects and therefore, they will be fixed first. (33).

3 IDENTIFYING PROBLEMS IN EVERYDAY WORK

This chapter focuses on identifying problems at the current state of software testing in the team that the author was working in. Solutions are also proposed at a high level.

3.1 Problems found in the use of subversion repository

The test scripts, which were, used to run automated tests are version controlled and stored in a subversion repository. All testers fetch these scripts to their personal computers and use them with their automated test tools. Because of the Agile software development and testing, even specifications can change during the time. Testing needs to adapt to these changes. Thus, it is not uncommon to make small changes to test scripts as the testing moves forward.

This can cause problems because the subversion repository is not up to date with the latest changes. Every tester might have totally different versions of test scripts in their personal computers and in the subversion repository. This may lead at least to the following problematic scenarios.

If a tester wants to test something, they have not tested before, they need to find the person who has the version of the test script with the latest changes as the newest version is not found in the subversion repository. This wastes a lot of testers' time.

Testing with outdated scripts may also lead to false negative results. Reporting a problem that does not actually exist wastes a lot of developer's and tester's time.

In a worst case scenario, the test scripts are lost if a tester's personal computer breaks.

3.1.1 Proposed solutions

The subversion repository needs to be kept up to date. This can be guaranteed with two steps. Firstly, all used test scripts should be updated from the subversion repository before every test. This ensures that the tester is not using older test scripts. Secondly, every time a tester makes changes to local test scripts, the tester should be reminded of updating these changes to the subversion repository after a certain period of time.

3.2 Problems found in used test sets

Test sets are a collection of tests, which are defined in a test database. The test database consists of a test plan, a test lab and test runs. The test plan describes the planned tests. It gives information about what is being tested, how and with what equipment. The test lab shows the tests and their states, for example, it shows if the test has passed or failed, what software build was tested and when. If failed tests exists, they will also have a fault report ID attached to them. Test runs show all the executed tests and information about them.

Tests are divided into three categories: Continuous Regression Testing (CRT), Continuous Integration Testing (CIT) and New Feature Testing. New Feature Testing is not in the scope of this thesis because it is something that is usually done manually before automating it as a part of regression testing.

There are also two software branches in use at a time, they use their own test sets. One of these branches is called Trunk. It is the mainline where developers commit their changes. The second one is called Branch, which is branched out from Trunk for a release. Branches live a short time before they are deleted. An example of Trunk and branching can be seen in figure 7 (34).

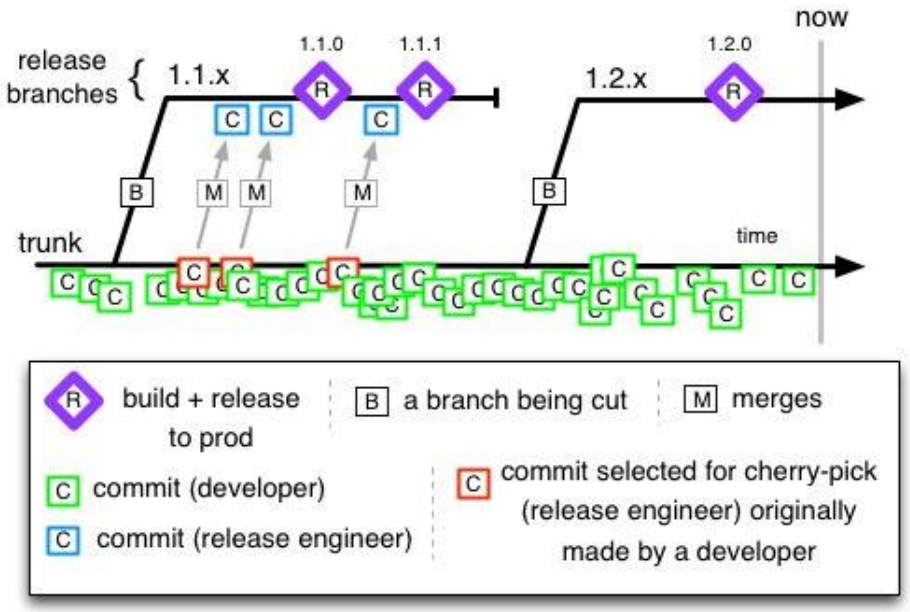


FIGURE 7. Example of Trunk and Branching (34).

Current automated testing tools run tests based on these test sets. The problem is that these sets have not been stored in the subversion repository at all. They have been newly created for every test environment due to issues that are better described in the chapter 2.3. This means that the test sets are more like a personal tool, than a tool that anyone can use.

3.2.1 Proposed solutions

A solution to this is to also store test sets in the subversion repository. For every CRT and CIT set, there would be a corresponding test set that anyone can use. This approach brings another issue but it is described more in depth in the next chapter.

3.3 Problems found in linking tests to a test database

Every test run in the company's automated test tool of the company is linked to a corresponding test found in the test database. This database includes all the tests and their status.

There is a problem importing test sets between test environments. The linking between the test and the test database is lost and they need to be added manually. This means that switching test environments adds unnecessary manual steps before testing can be started. This problem can also be seen in the same test environment if test sets are changed between Trunk and Branch.

3.3.1 Proposed solutions

Linking described in the earlier chapter should be part of test sets that are stored in the subversion repository. When a tester wants to test a certain test set, linking to a database should be created automatically.

3.4 Problems found in setting up a new test environment

All of the problems stated in the earlier chapters make setting up a new test environment a heavy task. Setting up a new test environment should be automated to suit the tester's needs. All the necessary tools, i.e. test scripts, test sets that need setting up, should be automated.

3.4.1 Proposed solutions

Tester should be able to choose what kind of test environment they want to be set up. Automation would then download test automation tools, scripts and test sets according to the tester's needs and set up the environment to be ready for testing without an input from the tester.

Benefits

Removing as much manual steps in everyday work benefits software testing greatly. More tests can be run in a shorter amount of time and the testers can focus on the most important things. Fully automated regression testing also ensures that possible broken legacy features are caught as soon as possible. Setting up a new test environment with just one click will greatly reduce the testing down time.

4 IMPLEMENTION OF SOLUTIONS

This chapter focuses on describing the implemented solutions. The implementation of proposed solutions is mostly done by using the Python-programming language.

4.1 Automating the use of subversion repository for test scripts

A new subversion repository was created to hold all the test scripts that were currently in use. The earlier repository held a lot of test scripts that have become obsolete. Either the functionality of software had changed so much that it had to be replaced with a new one or that the functionality was not tested anymore. A common problem was also found with the naming of test scripts. They were not simple and easily recognisable and often their naming was not in line with what was found in the test database. This could lead to confusion especially if someone unfamiliar with these tests was trying to make sense of the names.

Firstly, all the test scripts were renamed. Renaming was partly a collective effort where every tester gave their input to what kinds of naming rules should be applied. Once naming rules were agreed on, test scripts were renamed and committed to the new subversion repository.

Automating the usage of subversion repository started by implementing a Python library, which could check the used test scripts for updates, checking them out, checking for local modifications and checking how old these modifications are. For a subversion control, our team uses TortoiseSVN. It is a graphical interface for a subversion control which is integrated into the Windows shell. This means that it can be used from the file explorer window in the same way as any other file controlling feature. TortoiseSVN also comes with command line tools which the Python library uses to control the subversion repository.

Subversion, or Apache Subversion, is an open source version control system (VCS). There exists many tools which use this VCS. TortoiseSVN and SlikSVN are a few of them. The command line commands for controlling the subversion

are the same in most, if not all, of these tools. Therefore it does not matter what subversion control tool is being used because the Python script should work with them.

Next, it was time to create a test script which could be used together with the automation tools that our team uses. This script would use the earlier created Python library to handle the automatic check of used test scripts. The plan was to run this script before any other tests which would guarantee up-to-date test scripts during all tests. The script was finished and would “pass” or “fail” depending on if test scripts needed to be updated or if there were local modifications that were too old. The default time for too old modifications was set to 7 days but it can be changed if needed.

However, there is a design flaw in this kind of approach. The test automation tool loads all the wanted tests before starting to execute them. All the tests need to be finished or stopped before they can be loaded again. Because the script that checks for updates is part of the executable tests, even if the tests are updated, they will not affect in the current test execution. They will affect only in the next test execution.

4.2 Automating the use of subversion repository for test sets

As proposed in the chapter 2.2, test sets were also stored in the subversion repository. Test sets were made for each test plan found in the test database.

An automatic update of test set was achieved with the same script that updates test scripts. Both test scripts and test sets are stored in the same subversion repository essentially meaning that they are all updated at the same time. However, this was not enough to guarantee that the linking of tests to the test database works. This is discussed in more detail in the next chapter.

4.3 Automating the linking of tests to test database

To achieve an automated linking of tests to the test database, first it was needed to know how exactly this linking works and in what conditions it does not.

An automated test tool, which is used to run all our tests, saves the information of the current test environment into two files. These files include e.g. data the from current equipment connected to the test environment, the current test sets loaded into the environment and the information about what test is linked to what test in the test database. These two files were essential to automate the linking process.

One of these files is the same as the test sets that were created and stored in the subversion repository. The other file stores info about the connected equipment and test database linking. Both the test set file and equipment settings file store the test set name which is used as a link to the test database. However, the testers can choose this name themselves when they first create the linking in their environment. Every tester can use a different kind of naming convention for the test set name. Therefore importing test sets from another environment does not work in a new environment because the test set name must match the one found in the environments equipment settings.

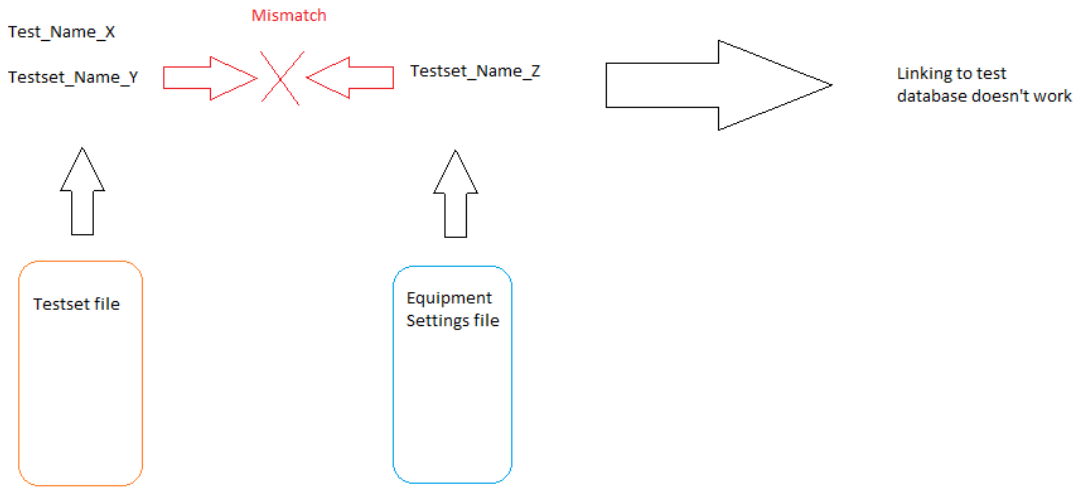


FIGURE 8. Testset name mismatch

A solution to this problem was to also store very basic equipment settings in the subversion repository which held the test set data that matches the data found in the test set settings. When a test set setting are fetched from the subversion

repository, the equipment settings are also fetched. This guarantees that the linking to the test database works.

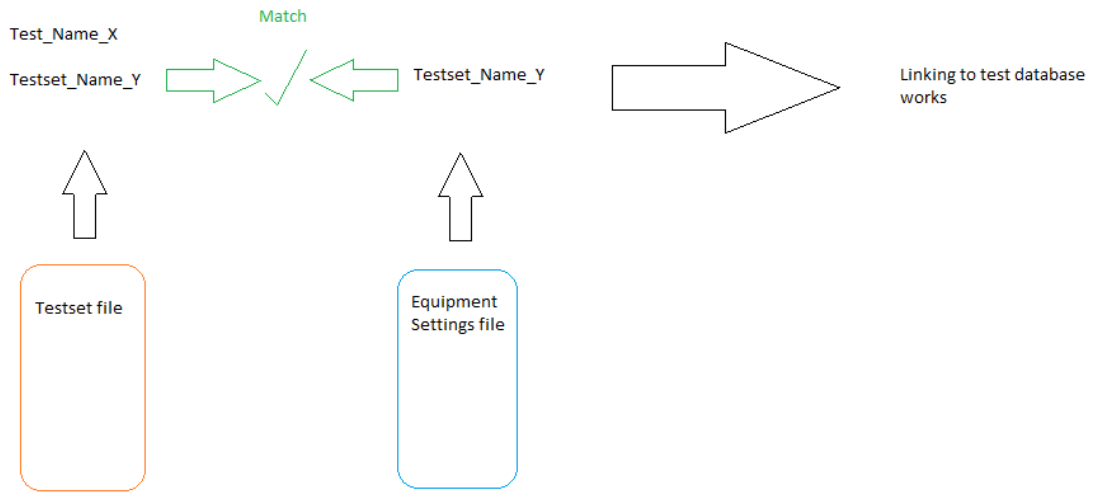


FIGURE 9. Testset name matches

Just replacing the settings with the ones found in the subversion repository would overwrite any and all important settings the test environment already has. For example, a test environment could have a signal analyser connected to it and it would have certain kinds of settings stored. To avoid losing these settings, a settings merger was created with the Python-programming language. This takes the existing settings and adds the settings from the subversion repository to it without losing anything in the process.

A backup feature was added just in case that something goes wrong and the environment is wanted to be restored back to what it was. Before the settings are merged, a backup file is taken and stored. This file holds the test set information and equipment settings from the environment. The same script, which set up the environment, can be called with an argument “undoChanges”. After this, the user can choose what backup they want to restore. The script was made to hold 25 backups but this can be easily changed if necessary.

4.4 Automating the setting up of a new test environment

Most of the work for automating the set up for a new test environment was already done in the earlier steps. All currently used tests were divided into test sets and stored in the subversion. Updating a test automation tool was implemented because it was essential for automating the linking between tests and the test database. The only thing needed was to make a script which brought all these functionalities together and made the steps from start to finish.

4.5 Cloudification

There are plans to have our current test automation tools in a cloud. This means that testers would be able to use any test environment that is connected to the cloud regardless where it is actually located.

All the usual benefits of cloud based applications apply here too. Cloudified test environments ensure that the hardware is utilized more. Instead of one tester using the environment for a maximum of 8 hours a day, multiple testers using the hardware can be using when needed, even throughout the whole day. Hardware or software problems can also be avoided. If a test environment in the cloud is malfunctioning, it is possible to use another one. With personal test environments, this usually is not possible and time would be wasted in troubleshooting the problem. Scalability is also better. Depending on the demand, new test environments can be added to the cloud for all the testers to use. (35, 36).

Some cloud test automation solutions already exist but they do not include the tool our team is using at the moment. This script, which sets up the test environment, can be used for the cloudification of this test automation tool. When this cloud test environment is built and our test automation tool has been installed, the script would be used to set up the environment with the test sets the tester wants.

5 CONCLUSION

The first part of the thesis focused on researching the theory of software testing and testing automation. While the process was familiar to me already through my earlier work with one of Nokia's testing teams, learning more about the theory behind it made the whole picture much clearer.

The aim of automating certain parts of everyday work was very clear as they were something I have been encountering in my daily work. Solutions to most of these problems were pretty straight forward too. The difficulty in linking the tests to the test database was the part that needed to be researched most. It was not known why the problem existed in the first place and finding it out took effort. The Cloudification part of the implementation was left undone for now because there are other things which need to be finished before it can be used.

The automated updating of test scripts did not turn out as well as intended. It was problematic to find out how the test automation tool works and how it truly prevents from updating the scripts before every test execution, at least with the current implementation method. This could be achieved by doing changes to the automation tool itself.

The objective of this thesis was to automate any manual tasks found in the testing process. This was achieved to a certain extent. Testers are now able to automatically set up their test environments and update their test scripts. Cloudification would automate this process even further in the future.

This thesis provided me increased knowledge of software testing, testing automation, programming in Python, using Subversion and the benefits of cloudification.

REFERENCES

1. Nokia. 2016.
   Date of retrieval: 25.10.2016
   http://company.nokia.com/en/about-us

2. Nokia. 2016.
   Date of retrieval: 25.10.2016
   https://networks.nokia.com/products/airscale-base-station

3. ISTQB Exam Certification. Why is software testing necessary?
   Date of retrieval: 18.10.2016
   http://istqbexamcertification.com/why-is-testing-necessary/

4. ISTQB Exam Certification. When do defects in software testing arise?
   Date of retrieval: 18.10.2016
   http://istqbexamcertification.com/when-do-defects-in-software-testing-arise/

5. ISTQB Exam Certification. What is independent testing? It is benefits and risks.
   Date of retrieval: 18.10.2016
   http://istqbexamcertification.com/what-is-independent-testing-its-benefits-and-risks/

6. ISTQB Exam Certification. What is the Psychology of testing?
   Date of retrieval: 18.10.2016
   http://istqbexamcertification.com/what-is-the-psychology-of-testing/#more-125

7. ISTQB Exam Certification. What is Software Testing Life Cycle (STLC)?
   Date of retrieval: 25.10.2016
   http://istqbexamcertification.com/what-is-software-testing-life-cycle-stlc/

8. Software Testing Course in Chandigarh. Webtech learning.
   Date of retrieval 21.11.2016
   http://www.webtechlearning.com/software-testing-course-chandigarh/

9. ISTQB Exam Certification. What are the principles of testing?
Date of retrieval: 25.10.2016
http://istqbexamcertification.com/what-are-the-principles-of-testing/

10. Ghahrai, Amir. 2008. Seven Principles of Software Testing
Date of retrieval: 25.10.2016
http://www.testingexcellence.com/seven-principles-of-software-testing/

11. Chapman, Alan. 2016. Pareto's principle. Businessballs.
Date of retrieval: 25.10.2016
http://www.businessballs.com/pareto-principle-80-20-rule.htm

12. ISTQB Exam Certification. What is Software Quality?
Date of retrieval: 18.10.2016
http://istqbexamcertification.com/what-is-software-quality/

13. ISTQB Exam Certification. What is the cost of defects in software testing?
Date of retrieval: 18.10.2016
http://istqbexamcertification.com/what-is-the-cost-of-defects-in-software-testing/

14. Soni, Mukesh. Defect Prevention: Reducing Costs and Enhancing Quality.
Date of retrieval: 18.10.2016
https://www.isixsigma.com/industries/software-it/defect-prevention-reducing-costs-and-enhancing-quality/

15. ISO Definitions. 1994. ISO 8402 1994.
Date of retrieval: 18.10.2016
https://www.scribd.com/doc/40047151/ISO-8402-1994-ISO-Definitions

16. ISTQB Exam Certification. What are Software Testing Levels?
Date of retrieval: 25.10.2016
http://istqbexamcertification.com/what-are-software-testing-levels/

17. Ambler, Scott W. 2014. Introduction to Test Driven Development (TDD). Agiledata.
Date of retrieval: 25.10.2016
http://agiledata.org/essays/tdd.html

18. ISTQB Exam Certification. What is Unit testing?
Date of retrieval: 25.10.2016
http://istqbexamcertification.com/what-is-unit-testing/

19. ISTQB Exam Certification. What is Integration testing?
Date of retrieval 9.11.2016
http://istqbexamcertification.com/what-is-integration-testing/

20. Overbaugh, John. 2007. How do testers do integration testing? What are top-down and bottom-up approaches in integration testing? TechTarget.
Date of retrieval 9.11.2016
http://searchsoftwarequality.techtarget.com/answer/How-to-do-integration-testing

21. ISTQB Exam Certification. What is Regression testing in software?
Date of retrieval 9.11.2016
http://istqbexamcertification.com/what-is-regression-testing-in-software/

22. Barber, Scott. 2007. Please tell me how to identify a regression test case? TechTarget.
Date of retrieval 9.11.2016
http://searchsoftwarequality.techtarget.com/answer/Automating-regression-test-cases

23. Goldsmith, Robin F. 2015. Does a tester actually need test cases? TechTarget.
Date of retrieval 9.11.2016
http://searchsoftwarequality.techtarget.com/answer/Does-a-tester-actually-need-test-cases

24. ISTQB Exam Certification. What is smoke testing? When to use it? Advantages and Disadvantages
Date of retrieval 9.11.2016
http://istqbexamcertification.com/what-is-smoke-testing-when-to-use-it-advantages-and-disadvantages-2/

25. ISTQB Exam Certification. What are the advantages or benefits of using testing tools?
Date of retrieval 9.11.2016
http://istqbexamcertification.com/what-are-the-advantages-or-benefits-of-using-testing-tools/

26. ISTQB Exam Certification. What are the different types of software testing tools?
Date of retrieval 9.11.2016
http://istqbexamcertification.com/what-are-the-different-types-of-software-testing-tools/

27. ISTQB Exam Certification. What are the risks or disadvantages of using the testing tools?
Date of retrieval 9.11.2016
http://istqbexamcertification.com/what-are-the-risks-or-disadvantages-of-using-the-testing-tools/

28. Pham, Vu. 2013. Software Testing Process & Trend. KMS Technology.
Date of retrieval 21.11.2016
http://www.slideshare.net/kmstechnology/software-testing-process-trend

29. Software Testing Fundamentals. Defect Report.
Date of retrieval 9.11.2016
http://softwaretestingfundamentals.com/defect-report/

30. Reichert, Amy E. 2016. Write software defect reports that get results, boost credibility. TechBeacon.
Date of retrieval 9.11.2016
http://techbeacon.com/write-software-defect-reports-get-results-boost-credibility

31. 2014. 10 Tips of Writing Efficient Defect Report. Software Testing Class.
Date of retrieval 9.11.2016
http://www.softwaretestingclass.com/10-tips-of-writing-efficient-defect-report/

32. Test Driven Development (TDD). Agile Testing Framework.
Date of retrieval 10.11.2016
http://www.agiletestingframework.com/atf/testing/test-driven-development-tdd/

33. ISTQB Exam Certification. What is a Defect Life Cycle or a Bug lifecycle in software testing?
Date of retrieval 10.11.2016
http://istqbexamcertification.com/what-is-a-defect-life-cycle/

34. Hammant, Paul. 2013. What is Trunk Based Development? Paul Hammant's blog.
Date of retrieval 18.11.2016
http://paulhammant.com/2013/04/05/what-is-trunk-based-development/

35. Jennings, Richi. 5 Financial Benefits of Moving to the Cloud, How SMBs and small office home office users can save money with cloud computing. Webroot.
Date of retrieval 18.11.2016
https://www.webroot.com/hk/en/business/resources/articles/cloud-computing/five-financial-benefits-of-moving-to-the-cloud

36. Coles, Cameron. 11 Advantages of Cloud Computing and How Your Business Can Benefit From Them. Skyhigh.
Date of retrieval 18.11.2016
https://www.skyhighnetworks.com/cloud-security-blog/11-advantages-of-cloud-computing-and-how-your-business-can-benefit-from-them/

37. Software Testing Genius. Defect Report and Its Sample Template.
Date of retrieval 22.11.2016
http://www.softwaretestinggenius.com/defect-report-and-its-sample-template

APPENDICES

Appendix 1 Defect Report (37).

Sample Template for Defect Report / Software Problem Report

Defect ID: (Required)	System generated
Author: (Required)	System generated
Release/Build # (Required)	Build where issue was discovered
Open Date: (Required)	System generated
Close Date: (Required)	System generated when QA closes
Problem Area: (Required)	Describe the problem area
Defect or Enhancement: (Required)	□ Defect (Default) □ Enhancement
Problem Title: (Required)	Short one-line description
Problem Description: (Required)	A precise problem description with screen shots, if possible
Current Environment: (Required)	E.g. Win95 / Oracle 4.0 NT
Other Environment(s):	E.g., WinNT / Oracle 4.0 NT
Defect Type: (Required)	□ Functionality (Default) □ Architectural □ Connectivity □ Consistency □ Database Integrity □ Documentation □ GUI □ Installation □ Memory □ Performance □ Security and Controls □ Standards and Conventions □ Stress □ Usability

Who Detected: (Required)	□ Quality Assurance (Default) □ External Customer □ Internal Customer □ Development
How Detected: (Required)	□ Testing (Default) □ Review □ Walkthrough □ JAD
Assigned To: (Required)	Individual assigned to investigate problem
Priority: (Required)	□ High (Default) □ Critical □ Medium □ Low
Severity: (Required)	□ High (Default) □ Critical □ Medium □ Low
Status: (Required)	□ Open (Default) □ Being Reviewed by Development □ Returned by Development □ Ready for Testing in the Next Build □ Closed (QA) □ Returned by (QA) □ Deferred to the Next Release
Status Description:	(Required when) Status = “Returned by Development,” “Ready for Testing in the Next Build”
Fixed by:	(Required when) Status = “Ready for Testing in the Next Build”
Planned Fix Build #:	(Required when) Status = “Ready for Testing in the Next Build”