Toni Riikonen

Jatkuvan integraation toteutus tukiasemaohjaimen hälytyksen ja elvytyksen hallinnalle

Tietotekniikan koulutusohjelma

2013

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

JATKUVAN INTEGRAATION TOTEUTUS TUKIASEMAOHJAIMEN HÄLYTYKSEN JA ELVYTYKSEN HALLINNALLE

Riikonen, Toni Mikael Antero
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Marraskuu 2013
Ohjaaja: Niemi, Juha
Sivumäärä: 28
Liitteitä:

Asiasanat: Jatkuva integraatio, Jenkins, Tukiasemaohjain, GSM

Tämän opinnäytetyön tarkoituksena oli toteuttaa jatkuva integraatio tukiasemaohjaimen hälytyksen ja elvytyksen hallinnan ohjelmalohkolle. Hälytyksen ja elvytyksen hallinnan ohjelmalohko toimii DX200 GSM matkapuhelinjärjestelmän tukiasemaohjaimessa.

Jatkuva integraatio projekti oli aloitettu asiakkaan pyynnöstä. Projekti oli päätetty toteuttaa Jenkinsillä. Jenkins suorittaisi koodin kääntämisen, moduulitestaukset ja ilmoittaisi mahdollisilta virheistä. Lisäksi saataisiin Jenkinsiin lisättyjen lisätoimintojen eli pluginien avulla muuta tietoa koodin mahdollisista epäkohdista. Jenkins voisi tehdä tätä jatkuvasti tasaisin väliajoin, eikä koodaajan tarvitse tehdä muuta kuin lisätä uusin koodi Jenkinsin ajettavaksi. Tämän jälkeen koodaaja olisi vapaa muihin tehtäviin tuloksia odotellessa tai ajon voisi pistää menemään läpi työajan ulkopuolella, jolloin tulokset olisivat valmiita seuraavana päivänä.

Hälytyksen ja elvytyksen hallinnan ohjelmalohkolle tehtiin niin kattava moduulitestaus kuin käytössä oleva testipenkki vain salli. Ohjelmalohkolle tehtiin myös toimiva projekti Jenkinsiin. Kyseinen projekti voidaan liittää muiden haluttujen ohjelmalohkojen kanssa Jenkinsin käyttöön varatulle koneelle, missä kukin ohjelmalohko ajetaan vuoronperään niiden uusimmilla koodeilla.

IMPLEMENTATION OF CONTINUOUS INTEGRATION FOR BASE STATION CONTROLLER'S ALARM AND RECOVERY CONTROL

Riikonen, Toni Mikael Antero
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Software Engineering
Month 2013
Supervisor: Niemi, Juha
Number of pages: 28
Appendices:

Keywords: Continuous integration, Jenkins, Base station controller, GSM

The purpose of this thesis was to implement continuous integration for base station controller's alarm and recovery control program block. Alarm and recovery control program block works in DX200 GSM mobilephonesystem's base station controller.

Continuous integration project had been started because of customer's proposal. Project had been decided to be executed with Jenkins. Jenkins would compile code, execute module testing and notify from possible errors. Also with extra features implemented in Jenkins a.k.a plugings, one could get extra information from possible faults in code. Jenkins could do this continuously with steady phasing and only thing programmer would have to do is insert latest code for Jenkins to run. After this programmer is would be free to do other tasks while waiting for results or he could make run execute outside working hours and then results would be ready next day.

As comprehensive as possible module testing was made for alarm and recovery control program within the limits of used test bench. Working Jenkins project was also made for program block. Project in question can be added with other program blocks to computer reserved for Jenkins, where each program block is run after each other with their latest codes.

SISÄLLYS

LYHENTEET JA KÄSITTEET

A-interface	A-rajapinta, BSC:n ja MSC:n välinen rajapinta
Abis-interface	Abis-rajapinta, BTS:n ja BSC:n välinen rajapinta
Air-interface	Ilmarajapinta, MS:n ja BTS:n välinen radiorajapinta
AUC	Authentication Center, autentikointikeskus
BC	Billing Center, laskutuskeskus
BCF	Base Station Control Function, tukiaseman ohjaus toiminta
BCSU	BSC Signalling Unit, BSC:n signalointi yksikkö
BG	Border Gateway, raja yhdyskäytävä
BSC	Base Station Controller, tukiasemaohjain
BSS	Base Station Subsystem, tukiasemajärjestelmä
BSSGP	Base Station System GPRS Protocol, BSS:n GPRS protokolla
BTS	Base Tranceiver Station, tukiasema
CEPT	Conférence Européenne des Postes et Télécommunications
CGF	Charging Gateway Functions, yhdyskäytävän laskutustoiminnot
CI	Continuous Integration, Jatkuva Integrointi
CLS	Clock and Synchronasation Unit, kello- ja synkronointiyksikkö
CPD	Copy/paste detector, kopioi/liitä havaitsija
CPU	Central processor unit, keskusyksikkö
DCN	Data Communications Network, datakommunikaatioverkko
DNS	Domain Name Server, nimipalvelin
DX200	Nokian kehittämä puhelinkeskusjärjestelmä

EDD	Enviroment Definition Directory, ympäristömäärittelyha kemisto "säkki"
EIR	Equipment Identity Register, laitetunnisterekisteri
ET	Exhange Terminal, keskuspääte
FDM	Frequency Division Multiplex, taajuudenjakokanavointi
FSC	Fixed Network Switching Centre, kiinteän verkon keskus
Gb-interface	Gb-rajapinta, BSS:n ja GPRS:n välinen rajapinta
GGSN	Gateway GPRS Support Node, yhdyskäytäväsolmu
GTP	GPRS Tunnel Protocol, GPRS:n tunnel protokolla
GSM	Global System for Mobile Communications, maaiman-laajuinen digitaalinen matkaviestintäjärjestelmä
HLR	Home Location Register, kotirekisteri
IMEI	International Mobile Equipment Identity, yksikäsitteinen laitetunniste
IMSI	International Mobile Equipment Identity, yksikäsitteinen tilaajatunniste
IP	Internet protocol, Internet protokolla
LAN	Local Area Network, Paikallisverkko
LI	Legal Interception, laillinen salakuuntelu
MCMU	Marker and Cellular Management Unit, kytkentämatriisi- ja radioresurrsienhallintayksikkö
MML	Man-machine language, ihminen-laite kieli
MS	Mobile Station, tilaajamoduulilla varustetu matkapuhelin tai muu vastaava päätelaite
MSC	Message Services Switching Center, matkapuhelinkeskus
MT	Module test, moduulitesti
NMS	Network Management Subsystem, verkonhallintajärjes-telmä
NSS	Network and Switching Subsystem, verkonkytkentäjär-jestelmä

O&M	Operations & Maintance interface, BSS:n ja NMS:n välinen rajapinta
OMU	Operations and Maintance Unit, käyttö- ja ylläpitoyksikkö
Perl	Practical Extraction and Report Language, käytännöllinen poiminnan ja raportoinnin kieli
PCU	Packet Control Unit, pakettihallintayksikkö
RAM	Random-access memory, keskusmuisti
SGSN	Serving GPRS Support Node, operointisolmu
SIM	Subscriber Identity Module, tilaajamoduuli
SVN	Subversion, versionhallintajärjestelmä
TC	Transcoder, transkooderi
TDM	Time Division Multiplex, aikajakokanavointi
TDMA	Time Division Multiple Access, aikajakokanavointiin perustuva kanavanvaraus
TRX	Tranceiver, lähetinvastaanotin
VLR	Visitor Location Register, vierasrekisteri
XML	Extensible Markup Language, merkintäkieli

1 JOHDANTO

Tieto Finland Oy oli aloittanut jatkuva integraatio projektin asiakkaan pyynnöstä. Projekti oli päätetty toteuttaa Jenkinsillä. Sen tarkoitus olisi parantaa työn tehokkuutta ja koodin laatua. Koodaajan ei tarvitsisi muuta kuin siirtää koodi versionhallintajärjestelmään, josta Jenkins kyseiset koodit noutaa. Jenkin kääntäisi koodin ja suorittaisi tarvittavat moduulitestit. Tämän jälkeen koodaaja olisi vapaa muihin tehtäviin tuloksia odotellessa tai ajon voisi pistää menemään läpi työajan ulkopuolella, jolloin tulokset olisivat valmiita seuraavana päivänä. Tuloksista hän näkisi onnistuiko käännös ja aiheuttiko koodimuutos virheen jossain toiminnallisuudessa. Lisäksi saataisiin Jenkinsiin lisättyjen lisätoimintojen eli pluginien avulla muuta tietoa koodin mahdollisista epäkohdista. Saataisiin aina automaattisesti selville onko koodissa esim. alustamattomia tai käyttämättömiä muuttujia, pituudeltaan rajat ylittäviä funktion ja proseduurin koodeja tai samankaltaisia koodiosia. Pituudeltaan rajat ylittävät funktiot ja proseduurin koodit olisi ehkä syytä paloitella pienemmiksi. Samankaltaiset koodiosat voivat olla turhia, kun jossain jo tehdään kyseinen asia

Opinnäytetyön alkaessa Jenkins pohja oli saatu valmiiksi. Nyt sitä alettiin ottamaan käyttöön tukiasemaohjaimen eri ohjelmalohkoille. Tarkoitus oli ottaa se käyttöön niiltä osin kuin oli mahdollista hälytyksen ja elvytyksen hallinnan ohjelmalohkolle ja todeta eri osien ja lisäosien toimivuus. Pääpainona oli moduulitestien tekeminen ja niiden muokkaaminen sellaisiksi, että Jenkin pystyy ajamaan ne. Hälytyksen ja elvytyksen hallinnan ohjelmalohkolle siis luotiin oma projektinsa Jenkinsiin, josta tehtiin ajo kelpoinen ja valmis liitettäväksi muiden ohjelmalohkojen kanssa Jenkinsin käyttöön varatulle tietokoneelle.

2 GSM-JÄRJESTELMÄ

2.1 Historia

1980-luvun alussa huomattiin, että Euroopan maissa käytettiin monia erilaisia yhteen sopimattomia matkapuhelinjärjestelmiä ja tarve telekommunikointi palveluille oli kasvanut huomattavasti. Tästä syystä CEPT (Conférence Européenne des Postes et Télécommunications) perusti ryhmän GSM (Groupe Spéciale Mobile) määrittelemään yhteisen mobiilijärjestelmän Länsi-Euroopalle. GSM lyhenteen tulkinta muuttui aikojen saatossa nykyiseen muotoonsa ”Global System for Mobile communications”. (SYSTRA 2000, 10.)

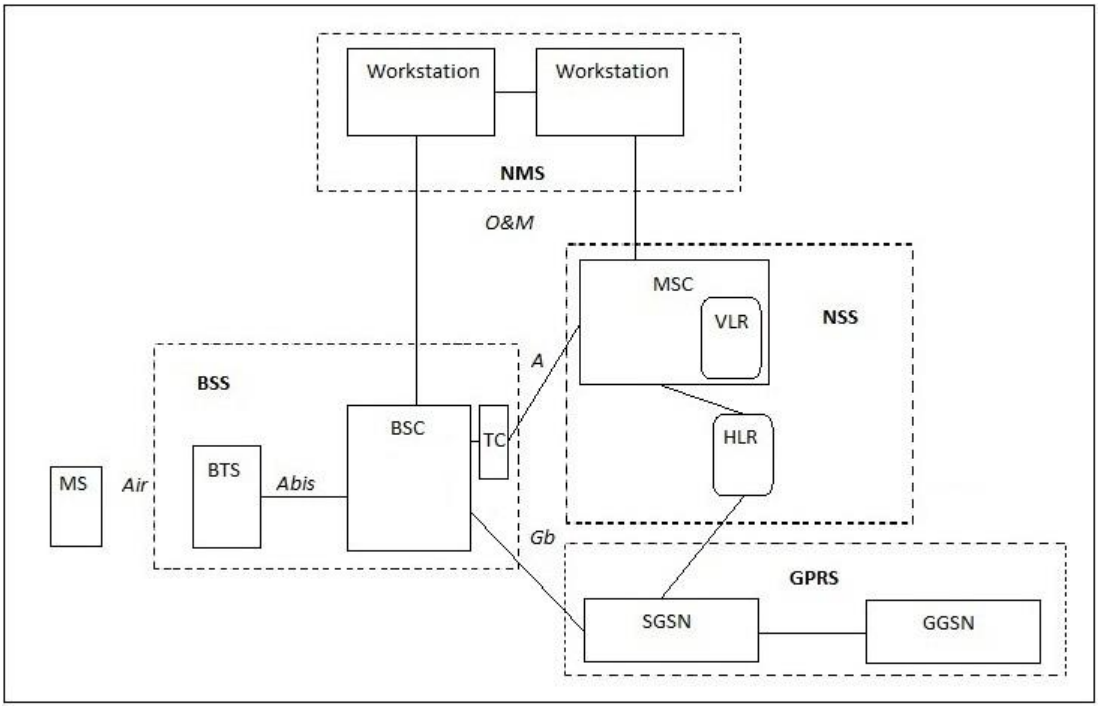
Vuonna 1991 soitettiin ensimmäinen virallinen GSM-puhelu ja vuonna 1992 maailman ensimmäinen GSM-verkko julkaistiin Suomessa. Myöhemmin GSM levisi myös Itä-Eurooppaan, Afrikkaan, Aasiaan ja Australiaan. Ainoastaan USA, valtaosa Etelä-Amerikasta ja Japani päättivät käyttää toisenlaisia järjestelmiä. USA:n PCS (Personal Communication System) tosin soveltaa GSM teknologiaa muutamin muutoksin. (SYSTRA 2000, 10, 12–13.)

GSM tunnetaan myös sukupolvitermillä 2G tai, jos se tukee GPRS:ää, 2.5G. Teknologian kehittymisen myötä nyt on olemassa myös UMTS-järjestelmä (3G) ja 4G.

2.2 Järjestelmän rakenne

GSM-järjestelmä voidaan jakaa seuraaviin osiin:

- Tukiasemajärjestelmä BSS (Base Station Subsystem)
- Verkonkytkentäjärjestelmä NSS (Netwrok Switching Subsystem)
- Verkonhallintajärjestelmä NMS (Netwrok Management Subsytem)
- Yleinen pakettiradiopalvelu GPRS (General Packet Radio Service)



Kuva 1. GSM-järjestelmän rakenne

2.2.1 Mobiiliasema MS

MS (Mobile Station) koostuu ensinnäkin laitteesta ME (Mobile Equipment), joka on matkapuhelin tai muu vastaava päätelaite. ME sisältää IMEI:n (International Mobile station Equipment Identifier) numeron. IMEI koostuu hyväksymiskoodista, valmiste-tunnisteesta ja valmistajan valitsemasta sarjanumerosta. (SYSTRA 2000, 16, 34, 81.)

Toiseksi MS:ään kuuluu tilaajatiedot sekä IMSI:n sisältävä erillinen moduuli SIM-kortti (Subscriber Identity Module). Tilaajatiedot ovat selvitettävissä SIM-kortissa sijaitsevan IMSI:n (International Mobile Subscriber Identity) avulla. IMSI koostuu maakoodista, verkkokoodista ja tilaajan tunnuksesta. (SYSTRA 2000, 16, 27.)

2.2.2 Tukiasemajärjestelmä BSS

BSS Sisältää:

- Tukiaseman BTS (Base Tranceiver Station)
- Tukiasemaohjaimen BSC (Base Station Controller)

- Transkooderin TC (Transcoder)

BTS:t sisältävät lähetinvastaanottimia TRX:iä (tranceiver), jotka muodostavat sijaintialueita. BTS huolehtii ilmarajapinnasta sekä sen signaloinnista, salauksesta ja puheen käsittelystä. Puheenkäsittelyllä tarkoittaen toimintoja virheettömän yhteyden takaamiseksi MS:n ja BTS:n välillä. BSC on keskus verkkoelementti BSS:ssä ja se ohjaa radioverkkoa. Se hallinnoi alaisuudessaan olevia BTS:iä BCF:ien (Base station Control Function) avulla ja vastaa yhteyden muodostamisesta MS:n ja NSS:n välillä, liikkuvuuden hallinnasta, tilastodatan keräämisestä sekä ilma- ja A-rajapinnan signalointien tukemisesta. TC kompressoi MS:stä tulevan puheen (16 Kbits/s) standardi nopeuteen puhelinverkossa (64 Kbits/s) ja päinvastoin. (SYSTRA 2000, 44, 71.)

2.2.3 Verkonkytkentäjärjestelmä NSS

NSS Sisältää:

- Matkapuhelinkeskuksen MSC (Mobile services Switching Centre)
- Vierasrekisterin VLR (Visitor Location Register)
- Kotirekisterin HLR (Home Location Register)
- Autentikointikeskuksen AUC (Authentication Centre)
- Laitetunnistusrekisterin EIR (Equipment Identity Register)

MSC vastaa puheluiden ohjaamisesta verkossa. Se tunnistaa puhelun alkamis- ja päättymispisteen sekä tyypin. Se voi myös toimia mobiili- ja lankaverkon välillä, jolloin sitä kutsutaan Gateway MSC:ksi. VLR on tavallisesti integroitu MSC:hen. Se pitää yllä ja päivittää MSC:n palvelualueella olevan tilaajan sijaintitietoja. Tiedot ovat väliaikaisia ja käyttäjän siirtyessä toiselle sijaintialueelle, data poistetaan vanhasta ja tallennetaan uuteen VLR:ään. HLR:ssä puolestaan on pysyvästi tilaajien puhelinnumerot MSISDN (Mobile Subscriber International ISDN Number) ja tilaajatunnistenumerot IMSI mutta HLR saa myös väliaikaisen tilaajan sijainnin VLR:ltä. AUC ja EIR sijaitsevat tavallisesti HLR:ssä. AUC autentikoi tilaajan yhdessä VLR:n kanssa käyttäen A3 algoritmia. EIR pitää yllä tietokantaa jossa IMEI numerot ovat joko valkoisella, harmaalla tai mustalla listalla. Valkoisella listalla ovat kaikki joissa ei ole mitään ongelmaa, harmaalla olevat ovat kyseenalaisia ja mustalla olevat on poistettu käytöstä. (SYSTRA 2000, 28, 40, 76, 81.)

2.2.4 Verkonhallintajärjestelmä NMS

NMS:n tarkoitus monitoroida verkon eri toimintoja ja elementtejä. Tätä varten on olemassa NMS/2000, joka koostuu työasemista, servereistä ja reitittimestä. Työasemat ovat yhteydessä tietokanta- ja tietoliikenneservereihin paikalliseverkon avulla LAN (Local Area Network) ja reitittimen kautta saadaan yhteys X.25-tietoliikenne verkkoon DCN (Data Communications Network), josta on yhteys GSM-verkkoon. Järjestelmää voidaan kutsua myös O&M:ksi (Operation and Maintance). NMS:n tehtäviin kuuluu: vianhallinta, kokoonpanohallinta ja toimintakyvynhallinta. (SYSTRA 2000, 73 - 75.)

2.2.5 Yleinen pakettiradiopalvelu GPRS

GPRS mahdollistaa pakettikytkentäiset yhteydet Internet-tyyppisiin palveluihin GSM-verkosta. Tiedonsiirtokapasiteetti varataan fyysisesti ainoastaan silloin, kun yhteydellä liikutetaan dataa. Toisin kuin piirikytkentäisessä, jossa vaaditaan yhteyden luonti ja ylläpito riippumatta siitä siirtääkö käyttäjä dataa vai ei. GPRS yhteydet ohjataan GPRS-runkoverkkoon BSC:n sisällä olevan paketinohjausyksikön PCU:n (Packet Control Unit) avulla. GPRS:n käyttö edellyttää GRPS:sää tukevan MS:n. (Penttinen, 2006, 158, 163.)

GPRS sisältää:

- Palvelevan GPRS-tukisolmun SGSN (Serving GPRS Support Node)
- GPRS-tukisolmu yhdyskäytävän GGSN (Gateway GPRS Support Node)

PCU:lla kytkeydytään SGSN:n, joka huolehtii GPRS-päätelaitteiden liikkuvuuden hallinnasta ja radiorajapinnansalauksesta. GGSN on GPRS-runkoverkon ja ulkopuolisten dataverkkojen välinen elementti, joka vastaa käytännössä reititintä palomuuri ominaisuuksilla, mutta osaa ohjata yhteyden aikana paketit oikealle SGSN:lle. Toiminnassa on mukana myös muita elementtejä. Verkkovierailu elementti BG (Border Gateway), joka muuttaa niiden välissä olevan verkon ei-julkiseksi. CGF (Charging Gateway Functions), joka huolehtii laskutuksesta. Lisäksi on vielä nimipalvelin DNS (Domain Name Server) sekä LI (Legal Interception), joka on viranomaisille

tarkoitettu liikenteen seuraamiseen soveltuva valvonta elementti. (Penttinen, 2006, 160 – 162.)

Käyttäjän lähettämää informaatiota ja oheistietoa siirrettäessä BSSGP (Base Station System GPRS Protocol) kuljettaa reititykseen ja palvelun tasoon liittyvän tiedon BSS:n ja SGSN:n välillä. GPRS-tunnelointiprotokolla GTP (GPRS Tunnel Protocol) ”paketoi” käyttäjän datan GPRS-ylläpitosolmujen ja runkoverkon välillä. (Penttinen, 2006, 164 – 165.)

2.3 Radioliikenne

Taulukko 1. GSM:n taajuusalueet. (Penttinen, 2006, 136.)

GSM-versio	**Taajuuskaista**	**Taajuudet (Uplink)**	**Taajuudet (Downlink)**
P-GSM 900	25 MHz	890 - 915 MHz	935 - 960 MHz
E-GSM 900	10 MHz	880 - 890 MHz	925 - 935 MHz
R-GSM 900	4 MHz	876 - 880 MHz	921 - 925 MHz
GSM 1800	75 MHz	1710 - 1785 MHz	1805 - 1880 MHz
GSM 1900	PCS 1900 -määrittelyjen mukainen	Riippuu alueesta	Riippuu alueesta

Tiedonsiirrossa on yhdistetty taajuudenkanavointi FDM (Frequency Division Multiplex) ja aikajakokanavointi TDM (Time Division Multiplex). Tämä on toteutettu niin, että verkon käytössä olevien taajuusalueiden sisällä tiedon siirto tapahtuu aikajakokanavoinnilla, jossa käytetään kahdeksaa aikaväliä. Aikavälit numeroidaan 0 – 7, ja yksi aikaväli T_i muodostaa oman kanavan. Kahdeksan aikaväliä muodostavat TDMA-kehyksen (Time Division Multiple Access), joissa kuljetetaan sekä dataa, puhetta että yhteyden ja verkon hallintaan tarvittavaa tietoa. Yksi kanava varaa yhden aikavälin jokaisesta TDMA-kehyksestä, joille siirrettävä datasanoma paloitellaan. (Grandlund, 2001, 130 – 131)

3 DX200 TUKIASEMAOHJAIN

3.1 DX200 tuoteperhe

DX200 on Nokian tuotenimi, jota käytetään yleisnimityksenä erillaisille tietoliikenneverkon muodostamiseen tarkoitetuille verkkoelementeille. Ensimmäinen DX 200 tuote oli kiinteän verkon puhelin keskus FSC (Fixed Network Switching Centre) mutta sen rinnalle on tullut myös muita, joista keskeisimmät ovat MSC, HLR ja BSC. Jokaisen perustana on periaatteessa samanlainen DX 200–laitteisto. Tämän päälle kootaan tarvittava ohjelmisto, josta muodostuu tiettyä tarkoitusta palveleva verkkoelementti, joka täyttää asetetut tiukat käytettävyys- ja reaaliaikavaatimukset. (Silander 1999, 2, 3, 48.)

3.2 Laitteistoarkkitehtuuri

DX 200 on periaatteessa yleiskäyttöinen tietokonejärjestelmä. Se voisi toimia missä tehtävässä tahansa mutta sisältää komponentteja, joita tarvitaan ainoastaan puhelinkeskus arkkitehtuurissa. Ohjelmiston avulla se saadaan toimimaan tiettynä osana. Osat ovat hyvin itsenäisiä mutta koska ne muodostavat kokonaisuuden, ne myös kommunikoivat sanomaväylän kautta. (Silander 1999, 48, 55.)

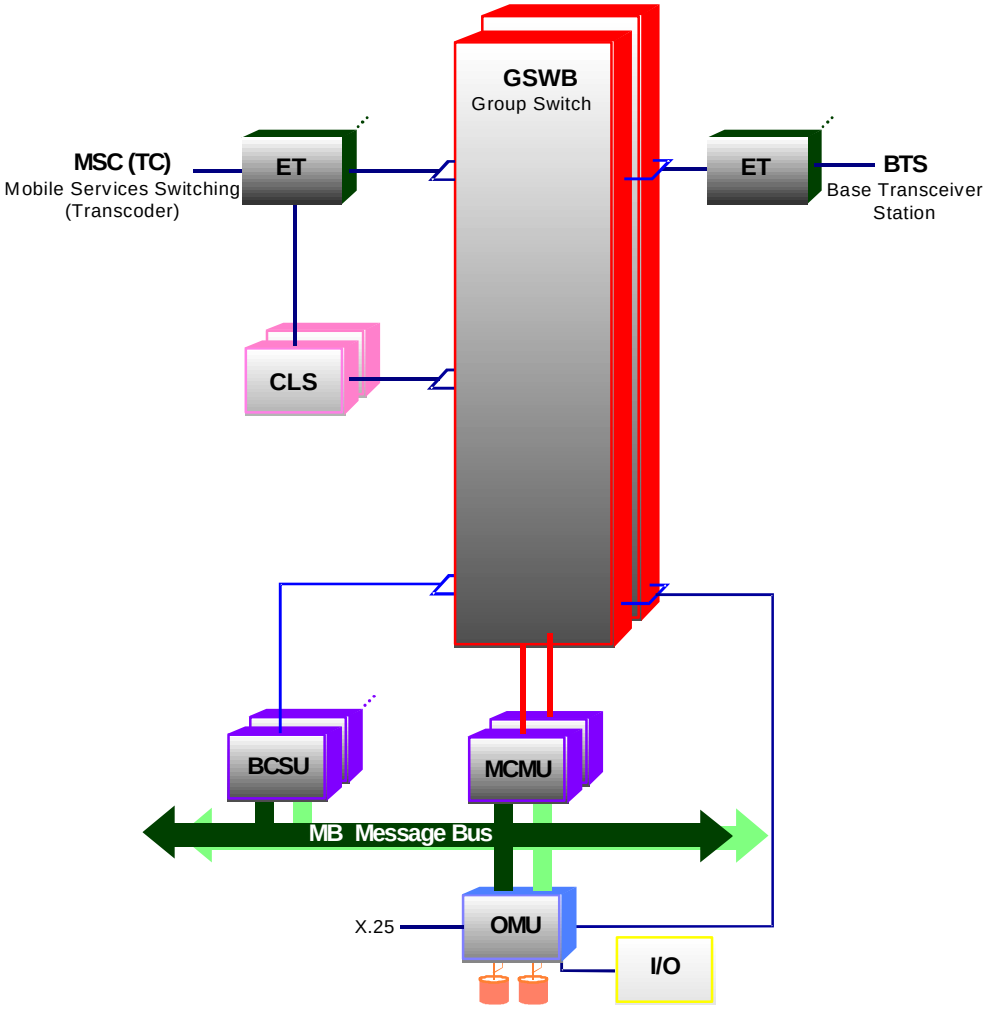
Järjestelmä on reaaliaikainen. Se vastaa palvelupyyntöihin tietyssä vasteajassa, joka riippuu järjestelmästä ja on oltava alle palvelulle asetetun aikarajan. DX 200 kuitenkin sisältää myös paljon sellaisia osia, joilla ei ole minkäänlaisia reaaliaikavaatimuksia. (Silander 1999, 66.)

3.3 Vikasietoisuus

DX 200 on vikasietoinen eli se kykenee jatkamaan toimintaansa laitteisto- tai ohjelmisto virheestä huolimatta. Silti virheistä, joihin järjestelmä kaatuu systemaattisesti, ei voida toipua. Toisin sanoen järjestelmä ei voi toimia, jos ohjelmoija on jättänyt ohjelmansa selvän virheen. Ensimmäinen varmennus menetelmä on 2N-varmennus,

jossa yksittäisellä resurssilla on tarkoin määritelty vararesurssi. Toinen varmennus menetelmä on N+1 –varmennus, joka voi olla korvaava tai täydentävä. Korvaavassa resurssilla on yksi tai useampi vararesurssi, joka voi korvata vikaantuneen. Täydentävässä puolestaan muut samassa tehtävässä olevat resurssit ottavat vikaantuneen tehtävän hoitoonsa. (Silander 1999, 67-68.)

3.4 BSC



Kuva 2. BSC:n rakenne (SYSTRA, 2000, 177)

BSC:n osat:

- Kytkentäkenttä GSWB
- Keskuspääte ET (Exchange Terminal)
- Kello- ja synkronointiyksikkö CLS (Clock and Synchronasation Unit)
- Paketinohjausyksikkö PCU (Packet Control Unit), joka ei näy kuvassa

- BSC:n signalointi yksikkö BCSU (BSC Signalling Unit)
- Kytkentämatriisi- ja radioverkkoresursienhallintayksikkö MCMU (Marker and Cellural Management Unit)
- Käyttö- ja ylläpitoyksikkö OMU (Operations and Maintance Unit)
- Sanomaväylä MB (Message Bus)

GSWB yhdistää inputin vaadittuun outputtiin. CLS generoi synkronnointi signaalit eri yksiköille. BCSU huolehtii SS7 signaloinnista MSC:n ja BSC:n välillä sekä LAP-D signaloinnista BSC:n ja BTS:n välillä. MCMU:n Marker hallitsee ja valvoo GSWB:tä ja Cellurar Management Unit vastaa soluista ja radiokanavista. OMU:n kautta henkilö voi suorittaa operorointi- ja huoltotehtäviä. ET:t sovittavat PCM linjat MSC:lle ja BTS:lle sopiviksi. PCU hallinnoi pakettidataa, joka ohjataan GPRS:ään. MB on yksiköiden välistä kommunikointia varten. (SYSTRA, 2000, 59, 180 – 185)

3.5 Ohjelmalohkot

DX-200 järjestelmä on modulaarinen eli ohjelmistopaketti on pilkottu pienempiin ohjelmalohkoihin, joilla jokaisella on oma tietty tehtävänsä. BSC on näin ollen jaettu osiin lohkomallin mukaisesti. Lohkomallissa on kolme tasoa: järjestelmälohko, palvelulohko ja ohjelmalohko. Lohkot ovat sisäkkäisiä, ts. BSC koostuu järjestelmälohkoista, jotka koostuvat palvelulohkoista, jotka koostuvat ohjelmalohkoista. (Silander 1999, 54 - 55, 59.)

Kommunikointi tapahtuu asynkronisin ja synkronisin palvelusanomin. Synkroninen palvelupyyntö pysäyttää sen pyytäjän toiminnan palvelunsuorittamisen ajaksi. Sitä käytetään funktiokirjastojen yhteydessä tai ajastinpalvelussa, jossa ajastin käynnistetään funktiokutsulla ja jäädään odottamaan laukeamissanomaa. Kahden eri ohjelmalohkon välinen sanomanvaihto mekanismi on asynkroninen, jossa palvelua pyytänyt asiakas ohjelma voi jatkaa toimintaansa heti palvelupyynnön jälkeen. Vastaus siirtyy sanomapuskuriin, josta se puretaan aikakajoin käsittelyyn. (Silander 1999, 63 - 65.)

Ohjelmiston määrittelyt yksittäisen ohjelmalohkon määrittelyjä lukuun ottamatta sijaitsevat ympäristömäärittelyhakemistossa EDD (Enviroment Definition Directory).

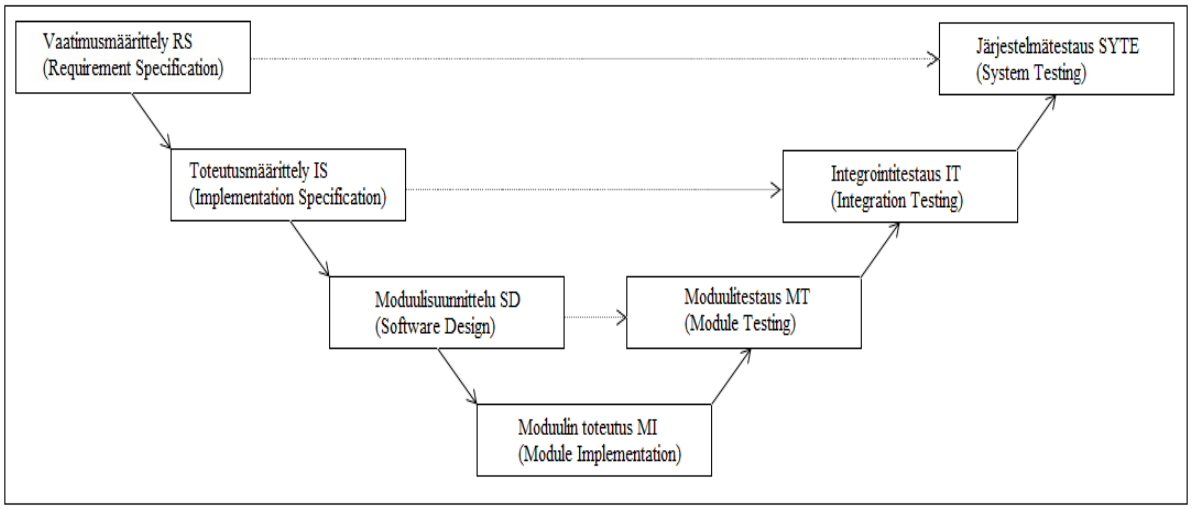
Se sisältää kaikki mahdolliset tietyllä hetkellä olemassa olevat määrittelyt ja on kaikkien ohjelmalohkojen käytettävissä. Ympäristömäärittelyhakemisto tunnetaan paremmin säkkinä.

4 OHJELMISTOTESTAUS

Ohjelmistotestaus on suunnitelmallista virheiden etsintää ohjelmaa tai sen osaa suorittamalla. Testauksen työvaiheisiin kuuluvat: testauksen suunnittelu, testiympäristönluonti, testin suorittaminen ja tulosten tarkastelu. (Haikala 2002, 281 - 282.)

4.1 Testausprosessin kuvaus

Ohjelmistoprojekti voidaan jakaa osiin testausta korostavan V-mallin mukaisesti.



Kuva 3. Ohjelmistotuotannon V-malli

Aluksi vaatimusmäärittelyssä kartoitetaan asiakasvaatimukset. Toteutusmäärittelyssä määritellään toteutettava järjestelmä ja ohjelmisto jaetaan moduuleihin. Moduulisuunnittelussa jokaisen moduulin sisäinen rakenne suunnitellaan, jotka toteutetaan moduulin toteutus vaiheessa. (Haikala 2002, 78, 81.)

Moduulitestauksessa testattavana on yksittäinen moduuli. Integrointitestauksessa moduulit ja moduuliryhmät yhdistetään ja tutkitaan niiden välisiä rajapintoja. Järjestelmätestauksessa tarkastellaan kokojärjestelmää ja tuloksia verrataan määrittelydokumentaatioon. (Haikala 2002, 287 - 288.)

4.2 Moduulitestaus

Moduulitestaus on siis ohjelmistokehitysprosessin testausvaiheista ensimmäinen. Yksittäistä moduulia verrataan moduulisuunnittelun ja arkkitehtuurisuunnittelun tuloksiin, tavallisesti tekniseen määrittelydokumenttiin. Testauksesta vastaa yleensä moduulin toteuttaja. Testauksen suorittamiseksi joudutaan usein toteuttamaan testipetejä (test bed) moduulin toimivuuden kokeilemiseksi. Testipetiin voi kuulua testiajureita (test driver) ja tynkämoduuleita (test stubs) ympäristöä simuloivina osina. Testiajurit mahdollistavat moduulin toteuttavien palveluiden kutsumisen ja tulosten tarkastelun. Tynkämoduulit korvaavat testattavan moduulin tarvitsemat muut moduulit. (Haikala 2002, 288.)

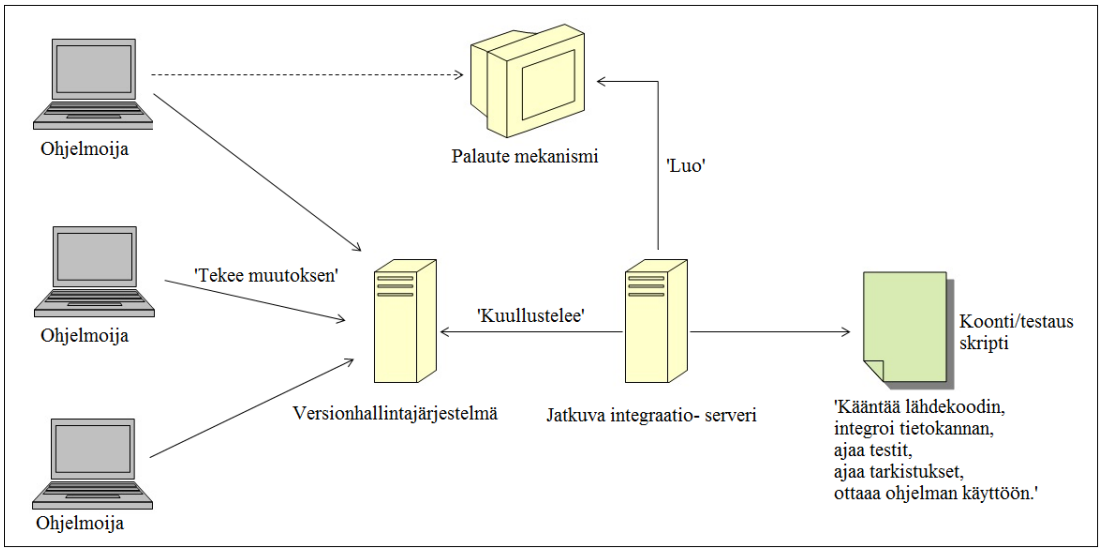
Virheiden korjaus on halvinta moduulitestaus tasolla, joten luonnollisesti tällä tasolla niiden löytäminen on suotavaa. Jos virheitä löytyy vasta ylemmillä tasoilla, täytyy sen korjaamisen jälkeen testiprosessi aloittaa alusta, koska virheiden korjaus voi aiheuttaa uusia virheitä. Näistä syistä moduulitestaus on automatisoitu. (Haikala 2002, 289.)

5 JATKUVA INTEGRAATIO

Jatkuva integraatio CI (Continuous integration) on ohjelmistokehitysmenetelmä. Termi on lähtöisin Extreme Programming kehitystyöstä. Jatkuvassa integraatiossa ryhmän jäsenet integroivat työnsä tavallisesti vähintään kerran päivässä. Integraatiot tarkistetaan automatisoidun käännöksen avulla, testit mukaan lukien. Tavoitteena olisi vähentää työhön liittyviä riskejä, kun tiedettäisiin jatkuvasti millä tasolla työ on ja virheiden löytäminen olisi helpompaa. (Fowler 2006.)

5.1 Käyttö

Prosessi alkaa uusimman toimivan version hakemisesta versionhallintajärjestelmästä (repository) työkoneelle. Seuraavaksi toteutetaan halutut muutokset, nämä voivat koskea koodi muutoksia, testien päivittämistä ja/tai uusien testien lisäämistä, jonka jälkeen mahdollisesti tarkistetaan muutetun ohjelmiston osion toimivuus testeillä omalla koneella. Sitten, jos joku on tallettanut tässä välissä uuden version versiohallintaan, päivitetään ja korjataan ensin oma koodi sen mukaan ennen kuin lähetetään oma versio versionhallintaan. Versionhallinnasta jatkuvaa integrointia suorittava työkalu hakee sen ja suorittaa koko ohjelmistolle työkalulle määritellyt toiminnot kuten kääntämisen ja testien ajamisen. Jos suoritus ei mene läpi, ohjelmistokehittäjälle ilmoitetaan siitä välittömästi. Näin voidaan havaita mahdolliset kehittäjän koneen käännöksessä ja testauksessa huomioimatta jääneet virheet ja ne saadaan nopeasti korjattua. (Fowler 2006.)



Kuva 4. Jatkuva integraatioprosessi

5.2 Käytänteet

Jatkuvassa Integroinnissa kannattaa noudattaa seuraavia käytänteitä.

Yksittäisen versiohallinnanylläpito:
Ohjelmisto projekteihin kuuluu paljon tiedostoja ja niiden tulee olla helposti saatavilla. Tätä varten tarvitaan versionhallintasysteemi, jonne tallennetaan kaikki mahdolliset tiedostot. Lisäksi vaikka versionhallinta järjestelmä tukisi ohjelmistojen jakamista eri haaroihin, tulisi niiden määrä pitää mahdollisimman vähäisenä. (Fowler 2006.)

Automaattinen käännös:
Ajan säästämiseksi kannattaa kääntämiseen liittyvät toiminnot automatisoida. Tämä olisi hyvä tehdä niin että koodi voidaan tarpeen tullen kääntää myös pienissä osissa. (Fowler 2006.)

Automaattiset testit:
Testien suorittaminen tulisi olla yksinkertaista. Niiden pitäisi auttaa löytämään sellaisia virheitä, joista pelkkä kääntäjä ei osaa ilmoittaa käännöksen yhteydessä. Tarvitaan siis testejä ohjelman loogisen toiminnan tarkkailemiseksi. (Fowler 2006.)

Päivittäiset muutokset:
Muokkaukset versionhallintaan tulisi tallettaa mahdollisimman usein, mieluiten vähintään kerran päivässä, jotta mahdolliset ristiriidat toisten ohjelmaosien kanssa löydettäisiin ja voitaisiin korjata nopeasti. Tutkittava alue on siis pienempi mitä useimmin versiohallintaan tehdään muutoksia ja säästetään taas aikaa. (Fowler 2006.)

Kaikkien versioiden integrointi:
Kehittäjät voivat kääntää ja testata ohjelman omalla koneellaan mutta käännösten toimivuus tulisi tarkistaa erillisellä integrointikoneella, jonka ympäristö on asetettu projektille sopivaksi, kehittäjien koneiden eroista johtuvien ongelmien minimoimiseksi. Integrointikoneen toiminnat voivat olla manuaalisia suorituksia tai automaattisia palvelimen suorittamia tehtäviä. Manuaalisessa suorituksessa kehittäjä tarkistaa itse integrointikoneelta, miten versiohallintaan tehtävät muutokset ja testit onnistuivat. Automaattisessa suorituksessa palvelin tarkkailee versonhallintaan tehtäviä muutoksia ja suorittaa automaattisesti muutoksen tapahtuessa käännöksen ja testit sekä muut mahdolliset tehtävät. (Fowler 2006.)

Nopea suoritus:
Tarkoitus on antaa kehittäjälle nopeasti palaute versionhallintaan tehdyistä muutoksista, eikä suoritus saa kestää kauan. Koska suurin hidaste löytyy käännöksen jälkeen tehtävistä testeistä, voidaan ne tarvittaessa jakaa pienempiin osiin. Päivityksen jälkeen ajettaisiin vain toiminnan varmistamisen kannalta välttämättömät testit ja loput ajettaisiin erikseen harvempaan tahtiin. (Fowler 2006.)

Oikeanlaisessa ympäristössä testaaminen:
Luotua ohjelmaa tulisi testata siinä ympäristössä, jossa sitä tullaan käyttämään sen valmistuttua. Muuten siihen saattaa jäädä huomiotta virheitä, jotka häiritsevät ohjelmaa sen lopullisessa ympäristössä. Jos samanlaisen testiympäristön luonti ei ole mahdollista, tulisi sen ainakin jäljitellä lopullista testiympäristöä niin paljon kuin mahdollista. (Fowler 2006.)

Informaation jakaminen:
Tuotettavan ohjelman viimeisin versio tulisi olla koko ajan kaikkien kehitykseen osallistuvien saatavilla. Näin on helpompi löytää mahdollisia virheitä ja puuttuvia pirteitä. Kehittäjien tulisi myös olla selvillä jokaisen käännöksen lopputuloksista ja tehdyistä muutoksista. (Fowler 2006.)

Automaattinen käyttöönotto:
Tarvittaessa voidaan koodi saattaa automaattisesti tuotantoon asti. Tällöin olisi syytä lisätä myös automaattinen palautus, siltä varalta että uudessa versiossa onkin jotain pielessä. (Fowler 2006.)

6 TYÖ

6.1 Käytetyt ohjelmat

Ohjelmien Java, Apache Ant, Python, Perl ja PMD sijainti kerrotaan Jenkinsille Windows ympäristömuuttujien kautta.

Jenkins:

Jenkins on sovellus, joka monitoroi toistuvien tehtävien suorittamista. Sillä on kaksi tehtävää. Kääntää ja testata ohjelmistoprojekteja toistuvasti eli toimia jatkuva integraatio systeeminä. Monitoroida ulkoisesti ajettujen tehtävien suorittamista, jotta virheen sattuessa siitä saadaan nopeasti tieto. (Jenkinsin www-sivut 2013)

Jenkinsiä käytetään selaimen kautta. Sille määritellään ensin renkikone yhteys. Tietokone, joka muodostaa yhteyden, toimii ajossa käytettävänä koneena. Ajon voi käynnistää menemään kyseisellä koneella, kun yhteys on muodostettu Javan avulla, projektin omasta välilehdestä. Ajon kulkua voi seurata ajon tiedoista, mistä myös näkee myös mahdollisen kaatumisen syyn. Projektin sivulla näkee viimeisimmän onnistuneen ajon tulokset ja muistissa olevien ajojen tuloksia voi tarkastella erikseen. Ajo näkyy suorituksen aikana välkkyvänä pallona projektin nimen ja ajon numeron vieressä. Punainen on epäonnistunut ajo, harmaa keskeytetty ja vihreä onnistunut.

Java Runtime Enviroment:

Käytetään Java ohjelmakoodin ajamiseen. Sen avulla muodostetaan ajoon käytettävästä koneesta yhteys Jenkinsiin. Sisältää Java Virtual Machinen, Java alustan ydinluokat ja tuet Java alustan kirjastoille. (Javan www-sivut 2013)

Apache Ant:

Apache Ant on Java kirjasto ja komentorivi työkalu, jonka tehtävä on ajaa rakenne tiedostoissa kohteiksi määritellyt prosessit ja toisistaan riippuvat jatkepisteet. Pääasiassa Antilla ajetaan Java sovelluksia mutta sitä voi käyttää myös muissa tapauksissa, kuten C tai C++ sovelluksissa. Käytetään Invoke Anttissa kutsuttavien xml tiedostojen ajamiseen. (Apache Antin www-sivut 2013)

Python:

Käytetään Python ohjelmointikielen ajamiseen. Python on tulkattu, vuorovaikutteinen, olioperustainen ohjelmointikieli. Se on sekoitus moduuleja, poikkeuksia, dynaamista kirjoittamista, erittäin korkean luokan dynaamisia datatyyppejä ja luokkia. Sillä on rajapinta moniin systeemi kutsuihin ja kirjastoihin, erilaisiin Windows systeemeihin ja se on laajennettavissa C tai C++ kielellä. Lisäksi se on käytettävissä laajennus kielenä, jos tarvitaan ohjelmoitava yhteys. Tarvitaan Apache Antin käynnistä-

essä ohjelmia ellei ohjelmaa käynnistetä Window batchilla ja ohjelmien tavitsemien tietojen, kuten testipenkkien IP-osotteiden, alustamiseen. Lisäksi se pitää huolen, ettei "makro-ohjelma" tai "moduulitestiohjelma" jää pyörimään liian pitkäksi aikaa, jos jokin menee pieleen. (Pythonin www-sivut 2013)

Perl:
Käytetään Perl ohjelmointikielen ajamiseen. Perl on korkean luokan ohjelmointikieli. Prosessointi, tiedosto ja teksti manipuloinnin ansiosta se soveltuu tehtäviin, jotka sisältävät nopeita prototyyppien tekemisiä, systeemi palveluja, ohjelmatyökaluja, systeemin hallintatehtäviä, tietokantojen avaamisia, graafista ohjelmointia, verkkotyöskentelyä tai verkko-ohjelmointia. Perl johtuu C ohjelmointi kielestä ja vähemmässä määrin muista työkaluista ja kielistä. Tarvitaan "moduulitestiohjelma" testien tulos tiedostojen löytämiseen "moduulitestiohjelman" kansiosta ja niiden muokkaamiseen xml muotoon, jota käytettään tulosten näyttämiseen Jenkinsissä. Jos PAC:ia ei ole MT kansiossa, Perl hakee halutun version Jenkinsille. (Perlin www-sivut 2013)

PMD:
PMD on lähdekoodi analysaattori. Se etsii yleisiä ohjelmointivirheitä kuten käyttämättömiä muuttujia, tyhjiä catch lohkoja, tarpeettomia objektien luonteja jne.. Lisäksi se sisältää Jenkins projektissa käytetyn CPD:n (Copy-paste-detector), joka etsii koodista samankaltaisuuksia, jotka voisivat olla tarpeettomia. Tuetut kielet ovat Java, C, C++, C#, PHP, Ruby, Fortran, JavaScrip mutta siihen voi liittää myös muita ohjelmointikieliä. (PMD:n www-sivut 2013)

"Makro-ohjelma":
"Makro-ohjelma" on makropohjainen testityökalu. Sitä käytetään makrojen ajamiseen ja MML yhteyden muodostamiseen. Lisäksi sen avulla siirretään image BSC:hen.

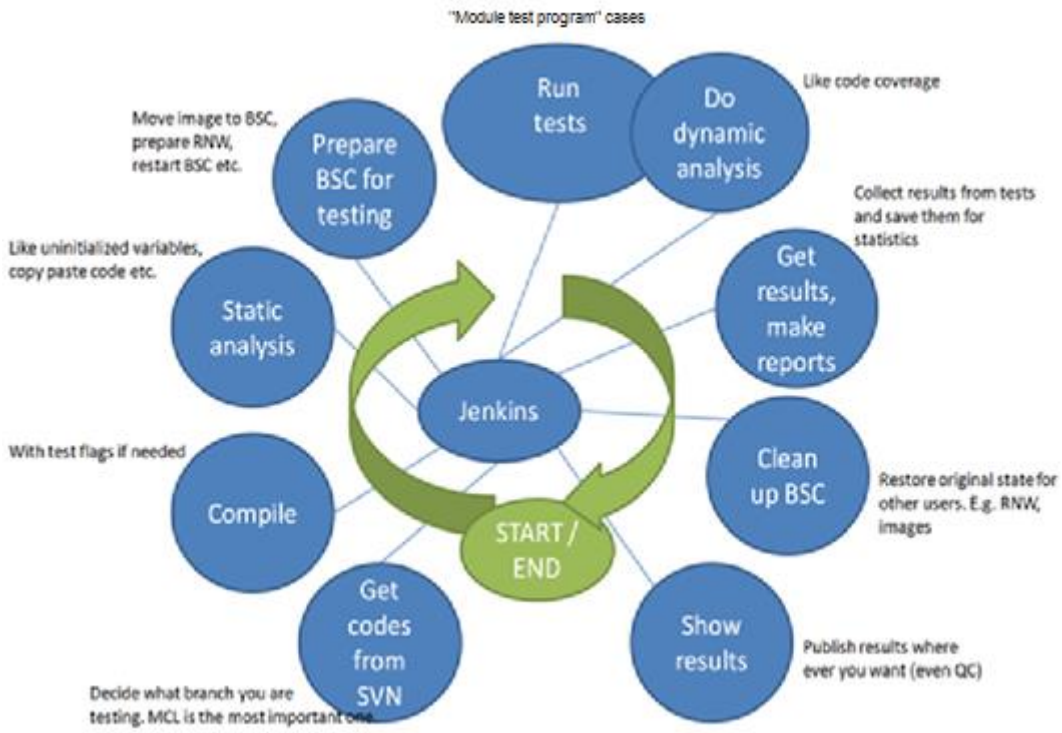
"Moduulitestiohjelma":
"Moduulitestiohjelma" on graafinen moduulitestityökalu. Sitä käytetään sillä toteutettujen moduulitestien ajamiseen sekä niiden ja niihin kuuluvien sanomien monitoroimiseen.

6.2 Toiminta

Projektin konfiguraatio alkaa projektin nimestä ja kuvauksesta. Sitten määritellään miten tuloksia säilytetään, eli kuinka kauan ja kuinka monta. Tämän jälkeen on ryhmä string parametreja. Tarvitaan moduulin "PAC" nimi, joka käännetään ja testataan sekä tieto siitä halutaanko käyttää uusinta vai jotain tiettyä versiota, jos PAC:ia ei ole MT kansiossa. PAC on tiedostopääte. Kyseinen tiedosto sisältää koodin ohjelmalohkon koodien kokoamiseksi sekä kääntämiseksi säkkiä vasten, jolloin saadaan tukiasemaan siirrettävä image. Sitten on kääntämisessä käytettävän säkin tiedot: sarja, versio, painos ja uudistus. BSC:n IP-osoite, minne koodit siirretään käännöksestä saadulla imagella ja missä testataan. Mitä imagelle tehdään: se siirretään BSC:hen mutta ensin otetaan varmuuskopio siellä jo olevasta imagesta, jos sellainen löytyy. Määritellään käytettävän imagen nimi.

Sitten on määritelty missä projekti ajetaan eli ajoon käytettävä tietokone yhteyden nimi. Valitaan luotavat kansiot ja niihin haettavat sisällöt. MT sisältää käännökseen tarvittavan PAC:n ja moduulitestauksessa käytettävän "moduulitestiohjelma" projektin ja sen vaatimat muut tiedostot sekä mahdolliset muut omat makrot, joita ei skripteissä ole. Pbuildiin tulee SVN:stä haettu koodi. SVN (Subversion) toimii versionhallintajärjestelmänä. Build sisältää aluksi projektissa käytettävät skriptit mutta sinne siirretään myös kaikki MT ja pbuild kansioiden sisältö ja Jenkins käyttää tätä kansiota ajon aikana.

Ennen suorituksen määrittelyä voidaan päättää suoritukselle erillinen laukaisin ja viilata suoritus ympäristöä, vaikkapa toteuta suoritus tietyin väliajoin ja keskeytä suoritus, jos se ei etene halutussa ajassa. Tämän jälkeen on varsinaisen suorituksen määrittely. Ensiksi kopioidaan MT kansion sisältö juureen. Sitten on ryhmä Invoke Anteja. Nämä kutsuvat Apache Antin avulla skriptejä, jotka puolestaan suorittavat haluttuja toimintoja. Toimii seuraavalla tavalla.



Kuva 5. Toimintakuvio

"Clean" tyhjentää log kansion, johon tulokset tallentuvat, ja juuren. Kopioidaan PAC build kansioon. "Compile" noutaa koodit SVN:stä määritetystä paikasta pbuildiin, mistä ne kopioidaa buildiin. "Compilen" tilalla oli tarkoitus olla "TNCov", joka olisi tarkkaillut kuinka ison alueen koodista moduulitesti kattaa mutta teknisistä syistä se jätettiin toteuttamatta. "Prepare" kääntää koodit PAC:n avulla "makro-ohjelmassa" aikaisemmin määriteltyä säkkiä vasten. Saatu image siirretään BSC:hen varmuuskopion ottamisen jälkeen ja BSC käynnistetään uudelleen, jotta ohjelmalohko ottaa käyttöön uudet koodit niistä tehdyllä imagella. BSC:n osien palattua takaisin toimintatilaan, ajetaan Windows batchin avulla makro "makro-ohjelmassa", jolla käynnistetään testiprosessit, eli ohjelmalohkot joita BSC:ssä ei ole. BCSU:t pakotetaan erikseen toiminta tilaan, koska niitä ei fyysisesti ole. Makro myös luo MT testissä käytettävän verkkoympäristön, tarvittavat BCF:t, BTS:t, TRX:t jne. ja muuttaa ne toiminta tilaan. ""Moduulitestiohjelma"" käynnistää "moduulitestiohjelman", jossa avataan MT kansioon määritelty projekti. Projektiin kuuluvat testit ajetaan MT kansioon määritellyn listan perusteella. Jos jokin testeistä menee pieleen tai kestää liian kauan, testattava ohjelmalohko käynnistetään uudelleen ja testi yritetään ajaa uudelleen. Mikäli se epäonnistuu vielä kerran, testi merkataan epäonnistuneeksi. "Static" käy

PAC:n avulla koodin läpi ja ottaa niistä TNCHECK ja SDLMA tulosteet. "Cpd" käy PMD:n avulla koodit läpi ja ilmoittaa toistensa kanssa samankaltaisia olevista koodipätkistä.

Seuraavaksi ovat suorituksen jälkeen tehtävät toiminnot. Määritellään mitkä tiedostot halutaan Jenkinsissä näkyviin: "moduulitestiohjelman" tekemän MT testauksen tulokset, joista nähdään mitkä testit menivät läpi ja mitkä eivät. Tämän jälkeen ajetaan "makro-ohjelmassa" makro, joka tyhjentää verkon kopioimalla BSC:n levyltä tallennuksen tyhjästä verkosta, jonka se ottaa käyttöön.

Viimeiseksi ovat Jenkinsissä näytettävät julkaisut. "Publish JUnit test result report" piirtää kaavion onnistuneista ja epäonnistuneista moduulitesteistä. Siinä näkyy kuinka mones ajo oli kyseessä ja suoritetuttujen testien määrä niin, että onnistuneet näkyvät vihreällä ja epäonnistuneet punaisella. "Publish TNCheck results" piirtää kaavion virheiden ja varoitusten määrästä. "Publish SDLMA report" piirtää kuvion SDLMA tuloksista. Siinä näkyy liian isot koodi palat punaisella ja rajojen sisällä olevat vihreällä. "Publish duplicade code analysis results" piirtää kuvion cpd tuloksista. Kaaviossa näkyy samankaltaisten koodipätkien määrä. Paljon samoja rivejä sisältävät punaisella ja melko paljon sisältävät keltaisella. Samoin kuin SDLMA:ssa rajat ovat määritelty koniguraatiossa.

7 TULOKSET

Hälytyksen ja elvytyksen hallinnan ohjelmalohkon projekti on nyt valmiina siirrettäväksi Jenkins koneelle. Kun se on tehty, Jenkins auttaa selvittämään aiheuttaako koodimuutos ongelmia toimintaan. Ohjelmoija voi vain tallentaa uuden koodin SVN:ään, jolloin se käännetään ja testataan automaattisesti. Tehokkuus paranee, kun aikaa ei kulu oikean moduulitestin etsimiseen ja manuaaliseen testaukseen. Käännös vaiheesta saadaan tietää kaatuuko koodin kääntäminen ja jos kyllä, niin mistä syystä. Moduuli testeistä saadaan selville aiheuttiko koodi muutos virheen koodin toiminnassa. Staattinen analyysi kertoo madollisista epäkohdista, kuten alustamattomista tai

käyttämättömistä muuttujista. SDLMA huomauttaa pituudeltaan rajat ylittävistä funktion ja proseduurin koodeista, jotka olisi ehkä syytä paloitella pienemmiksi. Cpd kertoo samankaltaisista koodiosista, jotka voivat olla turhia, kun jossain jo tehdään kyseinen asia. Näillä tiedoilla voidaan koodia sitten kehittää entisestään.

On kuitenkin huomioitava, että testipenkin rajoituksista johtuen moduulitesti ei ole koko koodin kattava. Koska kaikkia osia ei ole fyysisesti olemassa ja osa muista ohjelmalohkoista puuttuu kokonaan, joitakin testejä ei tällä hetkellä voi toteuttaa. Toivottavasti tulevaisuudessa käyttöön saadaan monipuolisempi testipenkki. Moduulitesti ei tietenkään myöskään kata uusia koodiin lisättäviä ominaisuuksia. Näille täytyy lisätä projektiin omat testit, joka onnistuu helposti päivittämällä MT kansiossa sijaitsevan "moduulitestiohjelma" projektin.

Valitettavasti vaikka "moduulitestiohjelma" ajaa koko projektin aina onnistuneesti läpi, kunhan koodi on kunnossa, se ei ole täysin stabiili, kun ajetaan monta testiä peräjälkeen. Tällöin ohjelmalohko joudutaan käynnistämään välillä uudelleen ja samoin täytyy tehdä myös muutamassa isommassa uudelleenkäynnistys-testissä. TNCov eli koodin kattavuustesti tallentaa tietoa ohjelmalohkon RAM muistiin. Koska ohjelmalohkon RAM muisti pyyhkiytyy uudelleenkäynnistyksen yhteydessä, TNCov jätettiin toteuttamatta.

LÄHTEET

Apache Antin www-sivut. 2013. Viitattu 15.11.2013. http://ant.apache.org/

Grandlund, K. 2001. Langaton tiedonsiirto. Porvoo: Docendo Finland Oy.

Haikala I. Marijärvi J. 2002. Ohjelmistotuotanti. Helsinki: Talentum.

Fowler, M. 2006. Continuous Integration. Viitattu 30.7.2013. http://www.martinfowler.com/articles/continuousIntegration.html

Javan www-sivut. 2013. Viitattu 15.11.2013 http://www.java.com/en/download/faq/whatis_java.xml

Jenkinsin www-sivut. 2013. Viitattu 15.11.2013. https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

Penttinen, J. 2006. Tietoliikenneverkot – Perusverkot ja GSM. Sanoma Pro Oy.

Perlin www-sivut. 2013. Viitattu 27.11.2013. http://learn.perl.org/faq/perlfaq1.html

PMD:n www-sivut. 2013. Viitattu 27.11.2013. http://pmd.sourceforge.net/

Pythonin www-sivut. 2013. Viitattu 27.11.2013. http://docs.python.org/2/faq/general.html

Silander, S. 1999. DX-AAPINEN Johdatus DX 200–ohjelmistotyöhön. Helsinki: Nokia Telecommunications Oy.

SYSTRA GSM System Training. 2000.