STADIA
HELSINGIN AMMATTIKORKEAKOULU

TEKNIIKAN JA LIIKENTEEN TOIMIALA

Tietotekniikka

Tietoliikennetekniikka

INSINÖÖRITYÖ

ATS3-TESTAUSJÄRJESTELMÄN EVALUOINTI

Työn tekijä: Maarit Kopra
Työn valvoja: Timo Kasurinen
Työn ohjaaja: Pekka Nuotio

Työ hyväksytty: 26.02.2007

Timo Kasurinen
lehtori

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ALKULAUSE

Tämä insinöörityö tehtiin Nokia Technology Platformsin S60 Mobile Runtime-yksikölle. Kiitän työni ohjaajaa insinööri Pekka Nuotiota sekä työni valvojaa lehtori Timo Kasurista avusta, rakenteellisesta palautteesta sekä neuvoista.

Helsingissä 26.2.2007

Maarit Kopra

INSINÖÖRITYÖN TIIVISTELMÄ

Tekijä: Maarit Kopra

Työn nimi: ATS3-testausjärjestelmän evaluointi.

Päivämäärä: 26.2.2007 Sivumäärä: 26 s.

Koulutusohjelma: Tietotekniikka Suuntautumisvaihtoehto: Tietoliikennetekniikka

Työn valvoja: Lehtori Timo Kasurinen

Työn ohjaaja: Insinööri Pekka Nuotio

Työ tehtiin Nokia Technology Platforms S60 Mobile Runtime-yksikölle. Työn tavoitteena oli evaluoida ATS3-testausjärjestelmä. ATS3-järjestelmällä voidaan automatisoida ohjelmistotestausta. Evaluoinnilla pyrittiin selvittämään voidaanko tuotekehitysvaiheessa oleva järjestelmä ottaa jo käyttöön vai vaatiiko se vielä lisäkehitystä.

Työssä kerrotaan aluksi yleisesti ohjelmistotestauksesta. Siinä käydään läpi myös testausprosessin vaiheet sekä kerrotaan lyhyesti avuksi kehitetyistä työkaluista. Työssä kerrotaan myös ATS3-järjestelmän edeltäjien ominaisuuksista, sekä miten ne eroavat tarkastelun kohteena olevasta järjestelmästä.

Varsinainen evaluointi alkoi tutustumalla ja asentamalla järjestelmä. Tämän jälkeen suoritettiin samat testit sekä ATS2.x-järjestelmässä että ATS3-järjestelmässä. Testauksen jälkeen analysoitiin testitulosten yhdenmukaisuuden perusteella, voidaanko siirtyä käyttämään ATS3-järjestelmää.

Testitulokset olivat käytännössä yhdenmukaiset suoritettaessa testejä matkapuhelimessa, mutta emulaattoritestaus täytyi jättää väliin. Lukuisista yrityksistä sekä järjestelmän kehittäjän avusta huolimatta testejä ei onnistuttu suorittamaan ATS3-järjestelmän emulaattoriympäristössä. Näin ollen päädyttiin tulokseen, ettei järjestelmää voida ottaa vielä käyttöön emulaattoritestauksessa, vaan se vaatii lisätutkimusta sekä perehtymistä järjestelmään.

Avainsanat: Ohjelmistotestaus, ATS3, ATS2.x, automatisointi.

ABSTRACT

Name: Maarit Kopra

Title: Evaluation of ATS3 Testing System

Date: February 26, 2007 Number of pages: 26 p.

Department: Information Technology Study Programme: Telecommunications

Instructor: Timo Kasurinen, Lecturer

Supervisor: Pekka Nuotio, B.Sc.

This graduate study was carried out for Nokia Technology Platforms S60 Mobile Runtime unit. The aim of the graduate study was to evaluate the ATS3 testing system. The ATS3 testing system enables the automation of software testing. The primary purpose of the evaluation was to determine whether the system which is currently in a development phase is ready for implementation or if it requires further development.

This study begins with an introduction to the basic principles of software testing. After that the testing process itself and the testing tools are presented. The features of earlier testing systems and their differences compared to the ATS3 testing system are described.

The actual evaluation was started by getting familiar with the system and installing it to a PC. After that the same tests were executed in both the ATS2.x system and ATS3 system. After the testing was completed the test results were compared to each other and based on found similarities an analysis was made whether it would be possible to transfer to using the ATS3 testing system.

The test results were practically similar when executing tests in a mobile device. However, emulator testing had to be skipped. Despite several attempts and guidance from the developer of the system, the tests could not be executed in the testing environment of the ATS3 testing system. The conclusion was that test system is not yet mature enough to be taken into use in emulator testing but further study and acquaintance with the system is needed.

Keywords: Software testing, ATS3, ATS2.x, automation.

SISÄLLYS

LYHENTEITÄ JA MÄÄRITELMIÄ

Apache Tomcat	Javalla toteutettu www-sovelluspalvelin.
API	Application Programming Interface. Sovellusrajapinta.
AppState	Application State. Sovelluksen tila.
ATS	Automatic Testing System. Järjestelmä, joka mahdollistaa testauksen automatisoinnin.
ASTE	Automatic System Test Engine. Järjestelmä, joka mahdollistaa käyttöliittymätestit.
BDF	Bitmap Description File. ASTE –järjestelmän käyttämä tiedosto.
Black Box Testing	Mustalaatikkotestaus. Funktionaalista testausta.
Bluetooth	Teknologia, joka mahdollistaa langattoman yhteyden mm. tietokoneiden, matkapuhelinten, digikameroiden ja muiden laitteiden välillä. Se perustuu virrankulutukseltaan pieneen radioyhteyteen.
Client	Asiakas, asiakaskone. Tietokone, joka voi hyödyntää toista samaan verkkoon kytkettyä tietokonetta (palvelinta) tiettyjen tehtävien suorittamisessa.
DAU-9T	Liitäntäkaapeli, joka mahdollistaa sarjaporttiyhteyden tietokoneen ja puhelimen välille.
DLL	Dynamic Link Library. Ositettu aliohjelmakirjasto. Tiedosto, joka sisältää ajettavaa ohjelmakoodia.
DKU-2	Liitäntäkaapeli, joka mahdollistaa USB -yhteyden tietokoneen ja puhelimen välillä.
Emulaattori	Laite, ohjelmoitava laite tai ohjelmoitavan laitteen ohjelma, jonka avulla tietokonejärjestelmä voi suorittaa toiselle järjestelmälle kirjoitettuja ohjelmia.
Flash Test Image	Puhelimeen asennettava tiedosto. Sisältää puhelimen perusohjelmiston.
FPS-10	Prommeri. Laite, jolla asennetaan ohjelmisto puhelimeen.
HTI	Harmonized Testing Environment. HTI huolehtii PC:n ja Symbian laitteen välisestä kommunikoinnista.
HW	Hardware. Laitteisto.
Java SDK	Java Software Development Kit. Java -kielen kehitysympäristö.
KW	Key Word. Avainsana ASTE järjestelmässä.

Localisation	Lokalisointi
MySQL	www -sovelluksissa käytettävä tietokanta
Prommer	Laite, jolla asennetaan ohjelmisto puhelimeen.
QC	Quality Center. Sovellus testaustietokantaan. Sisältää mm. testien kuvaukset ja testitulokset. Test Director (TD) sovelluksen uudempi versio.
Server	Serveri, palvelin
SIS tiedosto	Symbian Installation System. Asennustiedosto, jolla asennetaan halutut ohjelmistokomponentit Symbian käyttöjärjestelmää käyttävään puhelimeen.
STIF	Työkalun nimi. Testausrajapinta ei-käyttöliittymätesteille. Ei ole kirjainlyhennelmä.
Stub	Tynkä. Prosessi, joka toimintaympäristön yksinkertaistamiseksi korvaa oikean prosessiperheen ohjelmalohkon testauksessa, ja joskus myös toimintotestauksessa.
SUT	System Under Test. Testattava järjestelmä.
SW	Software. Ohjelmisto.
TCF	Test Case File. Tiedosto, joka sisältää testattavat testit sekä niiden testi ID:t, avainsanat ja parametrit. Testitiedosto.
TEM	Test Execution Management Tool. Erillinen Windows -sovellus, joka on linkitetty ASTE:n verkkolevyn kautta.
Test director	Sovellus testaustietokantaan.
Test harness	Testausrajapinta, protokolla, tapa testata.
Toolkit	Työkaluohjelmisto.
TQ	Test Quest. Kansio, joka sisältää ASTE -ohjelmiston bittikartat.
UI	User Interface. Käyttöliittymä. Ohjelmisto- ja oheislaitekokonaisuuden liitäntä, joka näkyy käyttäjälle.
White Box Testing	Lasilaatikkotestaus, valkolaatikkotestaus. Rakenteellista testausta.
(Test) worker	Tietokoneeseen asennettava sovellus, joka huolehtii testien suorittamisesta.

1 JOHDANTO

Ohjelmistotestaus kuuluu oleellisena osana ohjelmistotuotantoprosessiin. Huolellinen testaus voi viedä jopa 50 % käytettävistä resursseista. Ohjelmistotestaus pyrkii löytämään tuotteessa olevia virheitä. Lisäksi testauksen tarkoituksena on varmistaa, että kehitetty tuote täyttää sille prosessin alussa määritetyt vaatimukset. Sekä virheiden löytäminen että korjaaminen on tärkeää, koska ohjelmistovirhe voi aiheuttaa vammautumista, kuolemia, suuria taloudellisia menetyksiä tai vaikuttaa esimerkiksi käyttömukavuuteen. Kuuluisampia ohjelmistovirheitä, jotka aiheuttivat suuret taloudelliset tappiot, ovat Ariane 5-raketin räjähdys sekä Mars luotaimen menetys. Raketin räjähdys aiheutti n. 7,5 miljardin dollarin tappiot. [1; 2.]

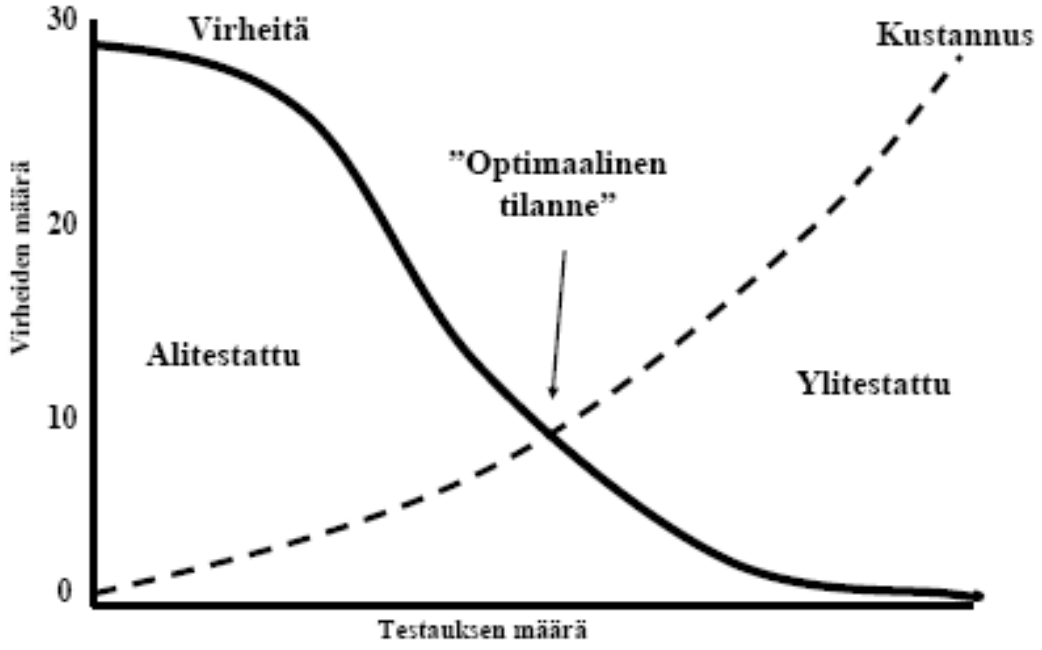
Koska tuotteen testaus vie huomattavasti aikaa, on avuksi kehitetty useita erilaisia testaustyökaluja. Tässä työssä kerrotaan yleisesti ohjelmistotestauksesta sekä sen avuksi kehitetystä ATS3-testausjärjestelmästä. Testattavat ohjelmistot on suunniteltu matkapuhelimiin. ATS3 on integroitu testausympäristö, joka mahdollistaa testien suorittamisen automaattisesti. Työssä käydään läpi ATS3:n edeltäjien (ATS2.x sekä ASTE) toimintaperiaate, kerrotaan ATS3:n rakenteesta sekä käydään läpi ATS3:n asennus, evaluointi ja tulosten analysointi.

Evaluoinnissa selvitetään täyttääkö ATS3 vaaditut ominaisuudet sekä se, onko ohjelmisto tarpeeksi kehittynyt ja vakaa, jotta voitaisiin siirtyä ATS2.x-järjestelmästä ATS3:seen. Evaluointi on tarpeen, koska ATS3 on vielä tuotekehitysvaiheessa. Evaluointi tapahtuu suorittamalla samat testit ATS3- sekä ATS2.25-järjestelmässä ja vertaamalla, ovatko tulokset samanlaiset.

2 YLEISTÄ OHJELMISTOTESTAUKSESTA

Jotta asiakkaalle voidaan toimittaa toimiva ja stabiili tuote, pitää se ensin testata. Markkinoille toimitettu tuote, joka ei toimikaan halutulla tavalla, voi aiheuttaa paljon lisäkustannuksia. Mitä kaikkea ja kuinka paljon sitten pitäisi testata? Koska kaikkea ei voi testata rajallisten resurssien takia, täytyy testaus suunnitella huolellisesti ja priorisoida tavoitteet. [3.]

Se kuinka paljon testataan, riippuu riskeistä. Riskejä ovat muun muassa markkinaosuuden ja luotettavuuden menetys, vioista aiheutuvat kustannukset sekä ylitestaus ja tehoton testaus. Koska testausaikaa on rajoitetusti, pitää päättää, mitä testataan ensimmäiseksi, mitä eniten ja mitä ei testata ollenkaan. [3.]



Kuva 1. Optimaalinen testauksen määrä. Kaikkien virheiden löytäminen on lähes mahdotonta ja tulee yleensä liian kalliiksi. [4.]

Jotta testauksesta saataisiin mahdollisimman paljon hyötyä, pitää testien olla tehokkaita, taloudellisia, kattavia sekä helposti ylläpidettäviä. Tehokkuudella tarkoitetaan tässä sitä, että testi löytää ohjelmistossa mahdollisesti esiintyvät virheet. Ylläpito ja muunneltavuus ovat myös tärkeitä, koska esimerkiksi puhelimissa on eroja. Erilainen näppäimistö, näyttöjen määrä ja muut tekniset ominaisuudet rajoittavat testien uudelleenkäyttöä. [3.]

Testien tehokkuutta ja kattavuutta voidaan selvittää kattavuustyökaluilla. Tällaiset analysaattorit mittaavat testikattavuutta erilaisilla kattavuusmitoilla. Yleisimmin käytössä olevia ovat:

- Lausekattavuus (Statement coverage). Ohjelman jokainen lause suoritetaan vähintään kerran.
- Päätöskattavuus (Decision coverage). Jokainen ehto saa testattaessa molemmat arvonsa.
- Ehtokattavuus (Condition coverage). Päätöksen kaikkien ehtojen on saatava kaikki arvonsa.
- Moniehtokattavuus (Multiple condition coverage). Testaus on suoritettava kaikkien ehtojen kaikilla yhdistelmillä.

Kattavuusmittojen paremmuusjärjestystä voi arvioida tutkimalla, täyttääkö tietyn mitan perusteella tehty testitapaus myös toisen mitan ehdot. Päätöskattavuudesta seuraa lausekattavuus, ja moniehtokattavuudesta puolestaan seuraa sekä ehto- että päätöskattavuus. Näin ollen moniehtokattavuus on vahvin tässä esitellyistä kattavuusmitoista. [5.]

Testejä suunniteltaessa kannattaa kiinnittää huomioita siihen, voidaanko testit automatisoida. Testien automatisointi mahdollistaa testattavan alueen kattavamman testauksen, koska tietokone pystyy suorittamaan huomattavasti enemmän testejä kuin yksittäinen testaaja. Valitettavasti kaikkia testejä ei voida kuitenkaan automatisoida. Osa testattavista toiminnoista vaatii testaajan apua, esimerkiksi liittämään jonkin lisälaitteen testattavaan tuotteeseen, poistamaan sen tai arvioimaan, näkyykö näytöllä oikeanlainen kuva. [3.]

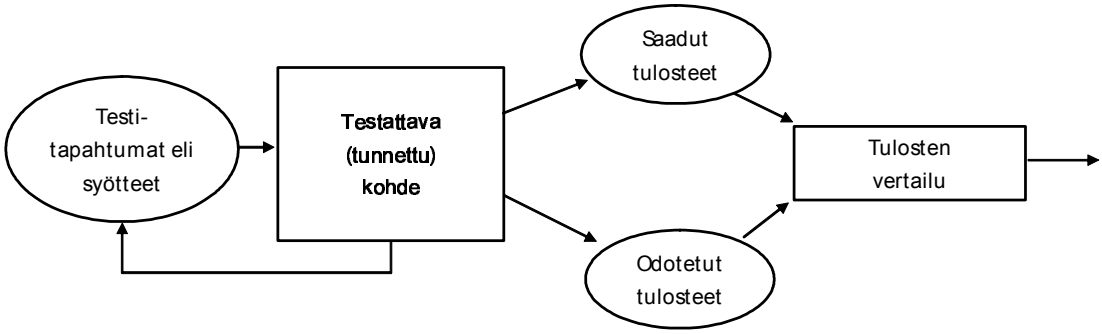
3 TESTAUSPROSESSI

Testausmuodot

Ohjelmistotestaus voidaan jakaa kahteen ”toimintatapaan”: lasilaatikkotestaukseen (White Box Testing) sekä mustalaatikkotestaukseen (Black Box Testing). Lasilaatikkotestaus on rakenteellista testausta, ja mustalaatikkotestaus on funktionaalista testausta. Rakenteellinen testaus tarkastelee ohjelman rakennetta kooditasolla ja vaatii käyttöönsä koko lähdekoodin. Sen avulla tarkastetaan, onko koodi hyvin rakennettua ja tehokasta.

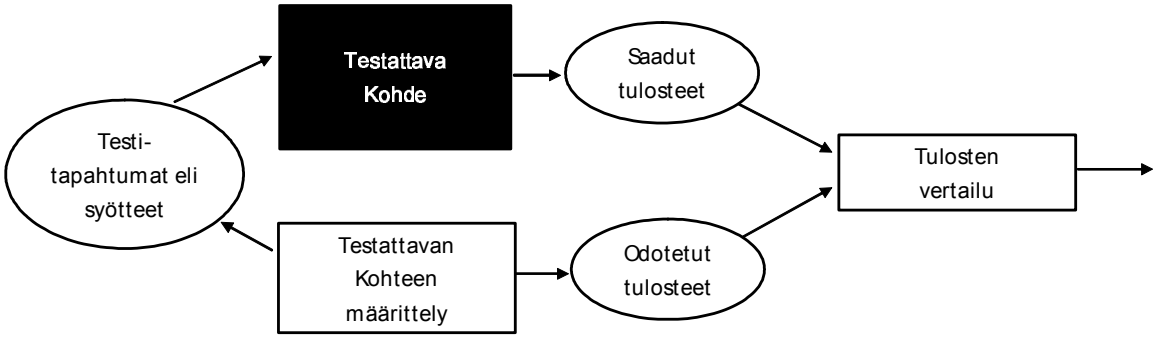
Monien tutkimusten mukaan edelleen yksittäinen tehokkain rakenteellinen testausmenetelmä on ohjelmakoodin suorittaminen rivi riviltä.

Yleensä lasilaatikkotestaus tehdään pienille ohjelman osille, esimerkiksi oliokielissä luokalle tai metodille. Lasilaatikkotestauksen menetelmiä käytetään lähinnä vain yksikkö- ja integrointitestauksessa. [6; 7.]



Kuva 2. White Box –testauksessa testitapaukset johdetaan kohteen, esimerkiksi koko ohjelman, sisäisestä rakenteesta ja logiikasta. [8.]

Mustalaatikkotestaus tarkastelee ohjelman kommunikointia muiden ohjelmamoduulien kanssa. Jos ohjelmamoduuli toimii ulospäin määritellyissä rajoissa, voidaan lähdekoodin tarkastelu jättää väliin. Funktionaalinen testaus sopii nykyiseen ohjelmistokehityksen tyyliin, joka on olio-suuntainen. Tällä menetelmällä ohjelman moduulit voidaan testata sellaisenaan, ja hitaampi lähdekoodin läpikäyminen käy turhaksi. Mustalaatikkotestauksessa etsitään tilanteita, joissa ohjelma ei toimi määrittelynsä mukaisesti. [6.]

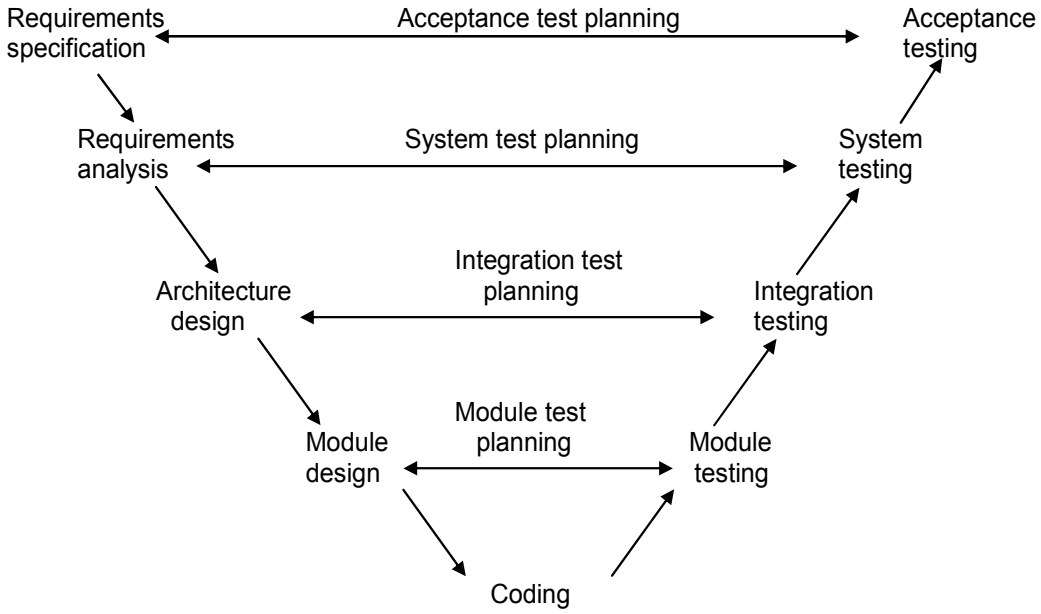


Kuva 3. Black Box -testauksessa testitapaukset johdetaan aina kohteen määrittelyn perusteella. [8.]

Testauksen tasot

Testaus voidaan jakaa useaan eri tasoon esimerkiksi käyttäen apuna V-mallia. V-malli on eräs variantti vesiputous- eli lineaarisesta mallista. Vesiputousmallin perusajatus on se, että ohjelmisto tuotetaan sarjana, joka sisältää eri vaiheita, ja jonka järjestys on ennalta määritelty. Eri vaiheet päättyvät tarkastuksiin ja katselmointeihin, ja jo suoritettuun vaiheeseen ei voida palata. Tyypillisiä vaiheita ovat

- esitutkimus
- järjestelmäsuunnittelu
- vaatimusmäärittely sekä toiminnallinen määrittely
- suunnittelu
- toteutus
- testaus
- käyttöönotto
- ylläpito.



Kuva 4. V-malli. [Lähdettä 9 mukaillen]

V-mallissa testaus on jaettu seuraaviin vaiheisiin:

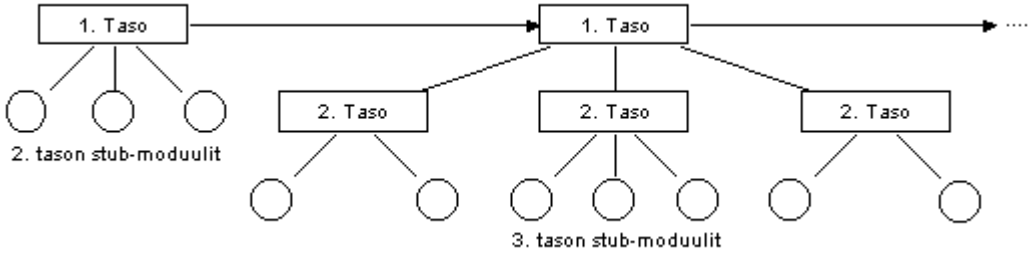
- moduulitestaukseen (module testing)
- integrointitestaukseen (integration testing)
- järjestelmätestaukseen (system testing)
- hyväksymistestaukseen (acceptance testing).

Moduuli- eli yksikkötestauksessa keskitytään yhden ohjelmamoduulin toiminnan tarkasteluun. Moduulin toimintaa verrataan moduulisuunnittelun ja arkkitehtuurisuunnittelun tuloksiin, kuten tekniseen määrittelydokumenttiin. Toisin kuin muissa testausvaiheissa, moduulitestauksen suorittaa yleensä moduulin toteuttaja. Moduulitestaus on yleensä White Box -tyyppistä testausta. [9; 10.]

Integrointitestaus aloitetaan, kun yksikkötestaus on suoritettu loppuun. Integrointitestauksessa yhdistellään moduuleita osajärjestelmiksi. Painopiste testauksessa on moduulien välisten rajapintojen toimivuuden tutkimisessa, ja testausten tuloksia verrataan tekniseen määrittelyyn. [9.]

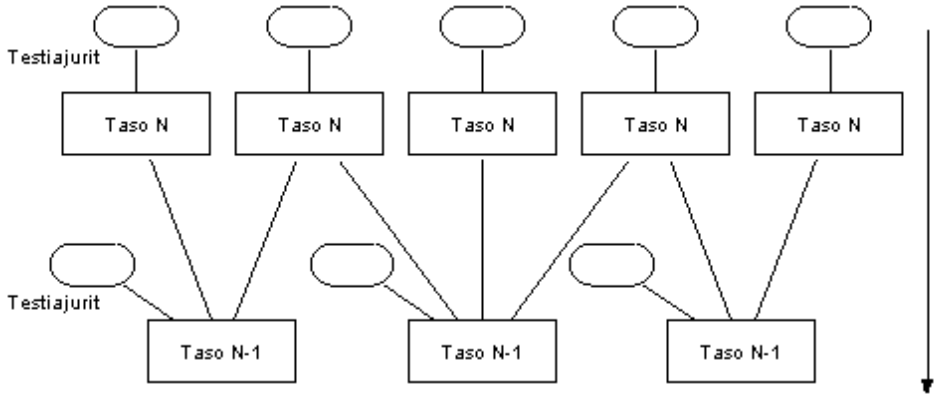
Tavallisesti integraatiotestauksen lähestymistapa on joko ylhäältä alas (top-down) tai alhaalta ylös (bottom-up). Top-down-lähestymistavassa testaus aloitetaan varsinaisesta pääohjelmasta, josta siirrytään kerros kerrokselta kohti rakenteen pohjalla sijaitsevia

moduuleja. Testauksessa tarvitaan tynkiä (stub) moduulinpätkiä, joita käytetään emuloimaan rakenteen pohjalla sijaitsevia ohjelmamoduuleja. Nämä stub-moduulit ovat hyvin yksinkertaisia, eivätkä sisällä mitään monimutkaisempaa toiminnallisuutta. Kun testaus etenee alemmalle tasolle, stub-moduulit korvataan oikeilla moduuleilla.



Kuva 5. Top-down-lähestymistapa integrointitestauksessa. [8.]

Bottom-up-lähestymistapa on käänteinen tapa verrattuna Top-down-menetelmään. Siinä testataan ensin rakenteen pohjalla olevat moduulit, ja vasta viimeisenä varsinainen pääohjelma. Stub-moduuleita Bottom-up-testauksessa vastaavat testiajurit. Testiajureilla voidaan suorittaa ja testata alemman tason komponentteja. Käytännössä useamman testiajurin sijaan luodaan yleensä testauksen mahdollistava ympäristö, joka toimii kaikkien moduulien yhteisenä testiajurina. [9; 10.]



Kuva 6. Bottom-up-lähestymistapa. [8.]

Järjestelmätestauksessa tarkastelun kohteena on koko järjestelmä, ja sen tuloksia verrataan ohjelmistojen toiminnalliseen määrittelyyn. Testaus aloitetaan, kun yksikkötestaus ja integrointitestaus on suoritettu. Järjestelmätestauksessa on tärkeää, että testaajina toimii kehitystyöstä mahdollisimman riippumattomia henkilöitä. Järjestelmätestauksen piiriin luetaan myös järjestelmän ei-toiminnallisten ominaisuuksien testaus, kuten kuormitus-, asennus- ja käytettävyystestit. [9.]

Jos määrittelydokumentaation vaatimukset on jäsennelty hyvin, on järjestelmätestaus helppo suunnitella vastaamaan näitä vaatimuksia. Muutoin vaatimukset voidaan ryhmitellä esimerkiksi näin:

- Volyymivaatimukset. Toimiiko järjestelmä oikein käytettäessä maksimivolyymiä tai yritettäessä ylittää se?
- Virhetilanteet. Toimiiko järjestelmä oikein, jos esimerkiksi tiedosto puuttuu tai se on tyhjä?
- Turvallisuus/ luotettavuus. Onnistuuko käyttäjä tuhoamaan tiedostoja tai muuttamaan tietoja, joita hänen ei pitäisi pystyä muuttamaan?
- Käytettävyys. Onko järjestelmä helppokäyttöinen ja selkeä?
- Palvelevuus. Antaako järjestelmä ne palvelut, joita määrittelyssä luvataan?
- Konfigurointitestaus. Toimiiko järjestelmä myös käyttäjän laiteympäristössä (mahd. erilainen kuin kehitysympäristö)? [4.]

Hyväksymistestauksen suorittavat ohjelmiston asiakkaat ja käyttäjät. Testauksen tarkoituksena on varmistaa, että ohjelmisto täyttää heidän vaatimuksensa. Hyväksymistestaus voidaan jakaa alfa- ja betatestaukseen. Alfatestauksen suorittavat todelliset käyttäjät kehitystyön tehneessä yrityksessä. Betatestauksen suorittavat asiakkaat omassa todellisessa käyttöympäristössään. [9.]

Edellä mainittujen vesiputousmallin ja sen varianttien ongelmia ovat mm. se, että liian suuria kokonaisuuksia käsitellään kerrallaan, iteratiivisuutta ei ole suunniteltu, malli on johdettu tuotannon organisoinnista teollisuudessa sekä eteneminen on sidottu tiukasti dokumentteihin, tarkastuksiin ja hyväksymisiin. Hyviä puolia ovat helppo omaksuminen, malli on yleisesti tunnettu, ja se tuottaa helposti johdettavan ja seurattavan ennakoitavan projektin. [9.]

Testausprosessin vaiheet

Testausprosessi voidaan jakaa vaiheisiin samaan tapaan kuin ohjelmistokehitysprosessi. Testaus alkaa suunnitteluvaiheella, jossa määritellään testauksen laajuus ja kriteerit, joilla testaus hyväksytään. Suunnitteludokumentin avulla kirjoitetaan testitapauksista koostuva testispesifikaatio. Suunnitteludokumentti ja testispesifikaatio voidaan myös tehdä yhdistettynä. Testausvaiheessa testispesifikaatiossa luetellut testitapaukset suoritetaan.

Testauksen suorituksesta laaditaan raportti, josta käy ilmi testauksen kattavuus sekä tiedot testauksen aikana löydetyistä virheistä. [5.]

4 OHJELMISTOTESTAUKSEN TYÖKALUJA

Ohjelmistotestauksessa työkalut jaetaan usein karkeasti sen mukaan, suoritetaanko testattavaa kohdetta testauksen aikana vai ei. Testauksessa käytetyt automatisoidut apuvälineet luokitellaan usein staattisen analyysin sekä dynaamisen analyysin ja testauksen työkaluihin. Kolmannen luokan muodostaa testauksen hallinnan työkalut. [8.]

Staattisen analyysin työkalujen tarkoitus on löytää virheitä ohjelman koodista ilman minkäänlaista ohjelman suoritusta. Esimerkiksi kääntäjä, joka koodin kääntämisen yhteydessä ilmoittaa koodista havaituista virheistä, on staattinen työkalu. Staattisia analysaattoreita käytetään yleisimmin ohjelmamittojen laskemiseen sekä ohjelman rakenteen tarkempaan analysointiin. Ohjelmamitoista voidaan päätellä ja ennustaa koodin mahdollisia ongelmakohtia, sekä analysoida ohjelman kompleksisuutta. Esimerkiksi CMT++, QA C/ C++/ Fortran ja PC-Metric ovat staattisen analyysin työkaluja. [8; 11.]

Jotta suoritusaikainen testaus ja sen analysointi olisi mahdollista, tarvitaan ohjelman pohjalle jonkinlainen ympäristö, joka mahdollistaa tulosten havainnoinnin. Tällainen ympäristö muodostetaan yleensä testiajureiden avulla. Testiajuri voi olla joko itsenäinen ohjelma tai testattavaan ohjelmaan yhdistetty osa. [8.]

Itse testaus suoritetaan yleensä joko käsin eli syötetään koko ajan testitapahtumia tai käytetään testiskriptiä, joka suorittaa tai syöttää testattavalle ohjelmalle skriptissä luetellut testitapahtumat. Testiskriptien etuna on testauksen nopeus ja helppous toistettaessa aikaisemmin suoritettuja testejä esim. regressiotestauksessa. Dynaamisessa testauksessa voidaan käyttää esimerkiksi simulaattoria. Se on työkalu, jolla simuloidaan suoritusympäristö ohjelman osille ja funktioille, joita sinällään ei voida suorittaa. Esimerkiksi Testwell Oy:n CTB –ohjelmistoa voidaan käyttää simuloinnissa. [8; 11.]

Testauksen kattavuuden ja laadun analysointiin voidaan käyttää esimerkiksi CTC++, QC/ Coverage, C-Cover ja Pure-Coverage ohjelmistoja. Suorituskykyä ja rasitussietoisuutta analysoivilla työkaluilla voidaan tarkastella ohjelman tehokkuutta, ja simuloida useiden käyttäjien ympäristöjä. Esimerkiksi SQA LoadTest- sekä PurePerformix - ohjelmistot sopivat tähän tarkoitukseen. [8; 11.]

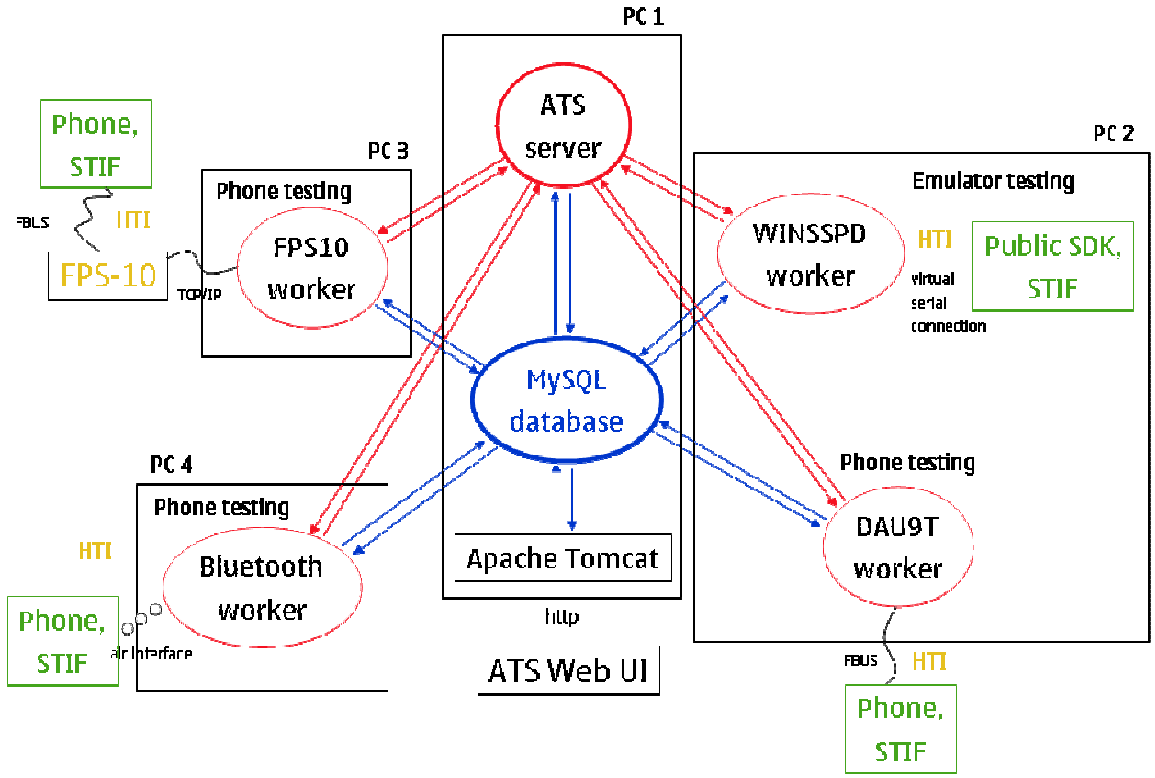
Dynaamisen analyysin työkaluihin luetaan kuuluvaksi usein myös testiaineistogeneraattorit. Näiden työkalujen tarkoituksena on automatisoida testiaineiston luontia. Testitapahtumat johdetaan usein suoraan ohjelman koodista saadun informaation perusteella, tai käyttämällä hyväksi valmiita tietokantoja. Esimerkiksi PiSCES Test Case Generator, STW/ Advisor sekä TESTBytes ohjelmistoja käytetään tässä tarkoituksessa. [8.]

5 ATS2.X, ASTE JA ATS3

ATS2.x

ATS2.x (Automatic Testing System) on testiautomaatiojärjestelmä, joka kehitettiin helpottamaan ja nopeuttamaan testausta. Järjestelmä muodostuu ATS palvelimesta ja vähintään yhdestä ATS client:ista (asiakas), joka voidaan haluttaessa asentaa samaan koneeseen kuin palvelin.

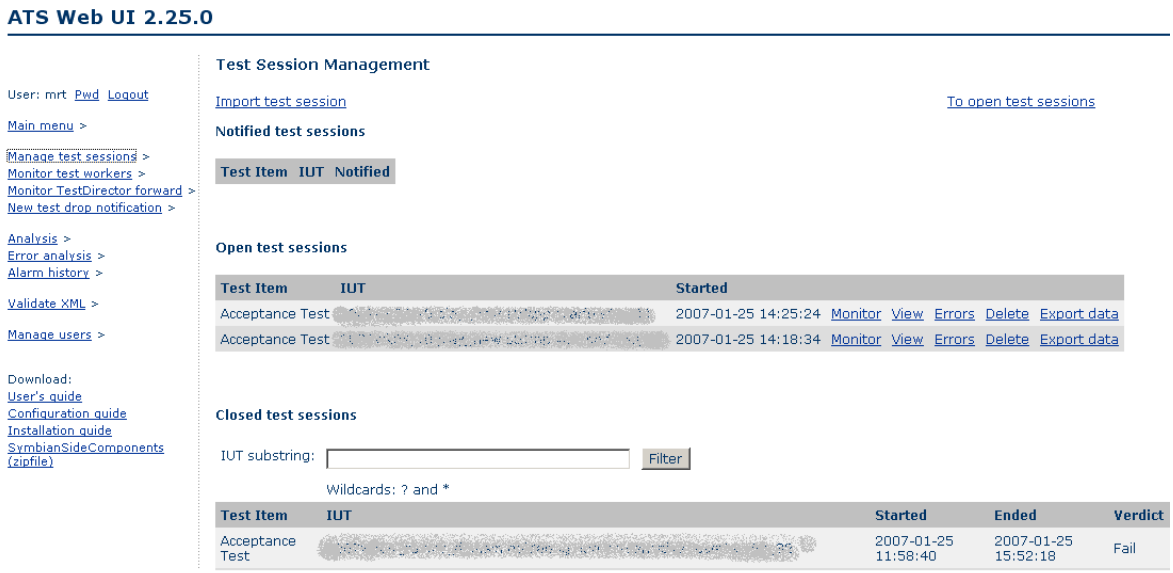
ATS2.x järjestelmä mahdollistaa testien automatisoinnin; ATS2.x asentaa flash test imagen, käynnistää puhelimen, kopioi tarpeelliset tiedostot, ja suorittaa määritetyt testit automaattisesti. Tämän lisäksi ATS2.x kerää ja tallentaa testitulokset suoritetuista testeistä, ja mahdollistaa raportoinnin sekä tulosten vertailun. ATS2.x:n selainpohjaisen käyttöliittymän avulla pystyy seuraamaan testien suoritusta reaaliaikaisesti millä tahansa internet yhteydellä varustetulla koneella. Varsinainen testiajo tapahtuu emulaattorissa tai puhelimessa.



Kuva 7. ATS2.x-arkkitehtuuri. [12.]

ATS2.x -ympäristön keskipisteenä on ATS-palvelin sekä tietokanta (kuva 7). Varsinainen testaus suoritetaan puhelimessa tai emulaattorissa, johon on asennettu STIF sekä HTI (Harmonized Testing Environment). Puhelin voidaan kytkeä workeriin esimerkiksi bluetoothin, FPS-10:n (Prommeri) tai DAU-9T-kaapelin avulla. Worker on eräänlainen testausmoduuli. Se on PC, johon on asennettu ATS-sovellus. Workerit kommunikoivat ATS palvelimen sekä tietokannan kanssa ja suorittavat testit.

ATS palvelin on Java-pohjainen PC-sovellus, joka hallinnoi testien suoritusta. Se jakaa testit worker-koneiden kesken, ja testien jälkeen kerää sekä tallentaa tulokset tietokantaan. ATS testitulokset on mahdollisista siirtää myös Quality Centeriin, mutta yhden testin siirtoaika on keskimäärin 10 sekuntia, joten useimmissa tapauksissa tämä ei ole järkevää.



Kuva 8. Näkymä ATS2.x:n internetliittymästä.

ASTE

ASTE (Automatic System Test Engine) on testausjärjestelmä, joka soveltuu UI-testaukseen (User Interface, käyttöliittymä). Käyttöliittymän testauksen tarkoituksena on paljastaa huonot vuorovaikutusmekanismit, ristiriitaisuudet sekä epäselvyydet käyttöliittymästä. UI:n ominaisuudet testataan siten, että varmistutaan ulkoasusääntöjen, estetiikan ja muun näkyvän sisällön välittyvän käyttäjälle virheettömästi.

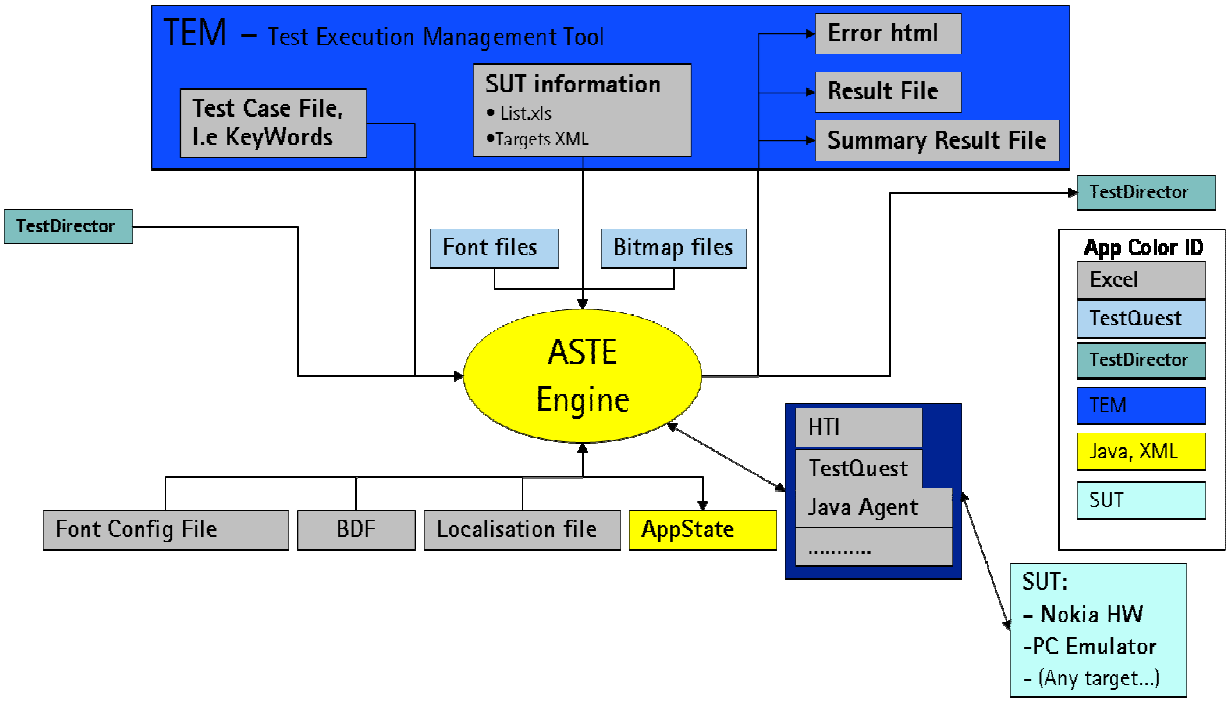
ASTE:n tärkeimpiä komponentteja ovat avainsanat (KeyWords, kw) ja Test Execution Management Tool -sovellus (TEM). ASTE-järjestelmä pystyy käsittelemään n. 15 avainsanaa. Nämä avainsanat on listattu testitiedostoon (Test Case File). Avainsanat käytännössä kuvaavat toimintoja, joita tehdään esimerkiksi käytettäessä matkapuhelinta. Avainsanoja käytetään muuntamaan manuaalitesti sellaiseen muotoon, jonka testaustyökalu ymmärtää. Tämän jälkeen testin voi suorittaa automaattisesti. [13.]

KeyWord	Function	Parameter

kw_PressHardKey	Press phone's button	Button name
kw_SelectMenu	Select menu item	Menu item's logical name
vw_VerifyText	Check if text is on the screen	Logical name for the text
…		

Taulukko 1. Esimerkki taulukosta, johon on listattu muutamia avainsanoja.

Test Execution Management Tool on Windows-sovellus, jolla hallitaan testausta. Sen avulla voidaan lisätä, poistaa, keskeyttää ja siirtää testitiedoston suoritusta. TEM-sovelluksen avulla voidaan myös suorittaa yksittäinen testi. [13.]



Kuva 9. ASTE 1.0 -järjestelmäkomponentit. [13.]

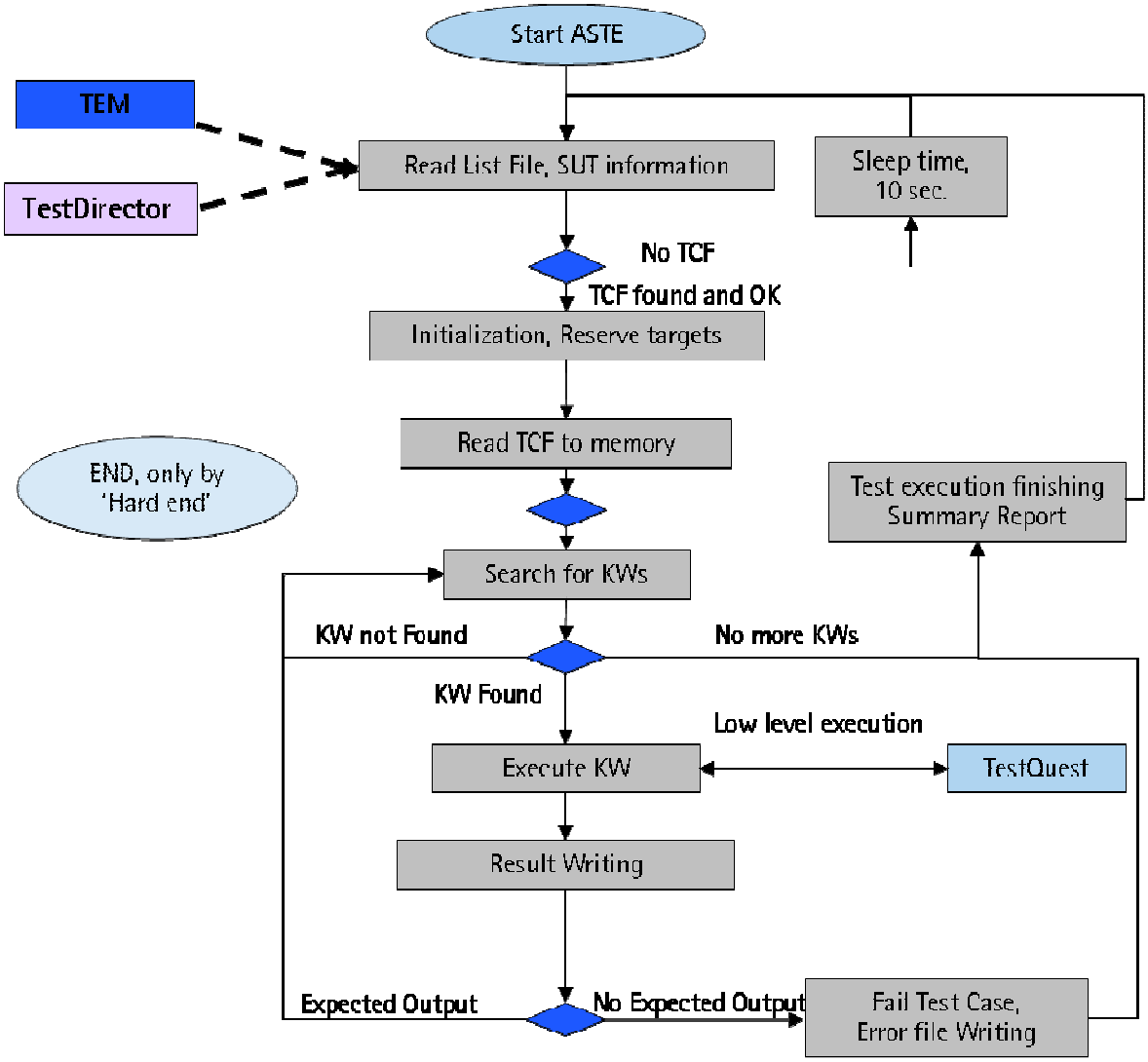
AppState eli Application State on eräänlainen testin esitila. Se nopeuttaa testausta, koska useilla testeillä on sama aloituspiste, mistä testaus voidaan aloittaa. AppState-komponentti mahdollistaa tehokkaan testauksen. Sen avulla testeistä tulee itsenäisempiä. Tämän avulla voidaan ajaa testit missä tahansa järjestyksessä. [13.]

Localisation eli lokalisointi hyödyntää SUT-komponentin (System Under Test) tietoja. Siitä selviää mm. testattava kielivariantti ja laite. [13.]

Taulukko 2. Esimerkki lokalisointitiedostosta.

Logical name	**English**	**Finnish**	**…**
§Phonebook§	Contacts	Osoitekirja	…
§Exit§	Exit	Poistu	…
§Create_Message§	Create new message	Luo uusi viesti	…

BDF-tiedosto (Bitmap Description File) toimii navigointiapuna testitiedoston ja varsinaisen bittikartan välillä, joka on sijoitettu TQ-kansioon (Test Quest). Bittikarttoja tarvitaan kuvaamaan erilaisia kenttiä/ painikkeita, joita ei voida kuvata loogisilla nimillä. Käytännössä BDF-tiedostoon syötetään parametri, SW ja HW, ja ulostulona saadaan mitä bittikarttaa tarvitaan ja missä se sijaitsee. Jokainen kielivariantti vaatii oman BDF-tiedoston. [13.]



Kuva 10. ASTE System Engine. Kaaviosta selviää miten ASTE suorittaa testit. [13.]

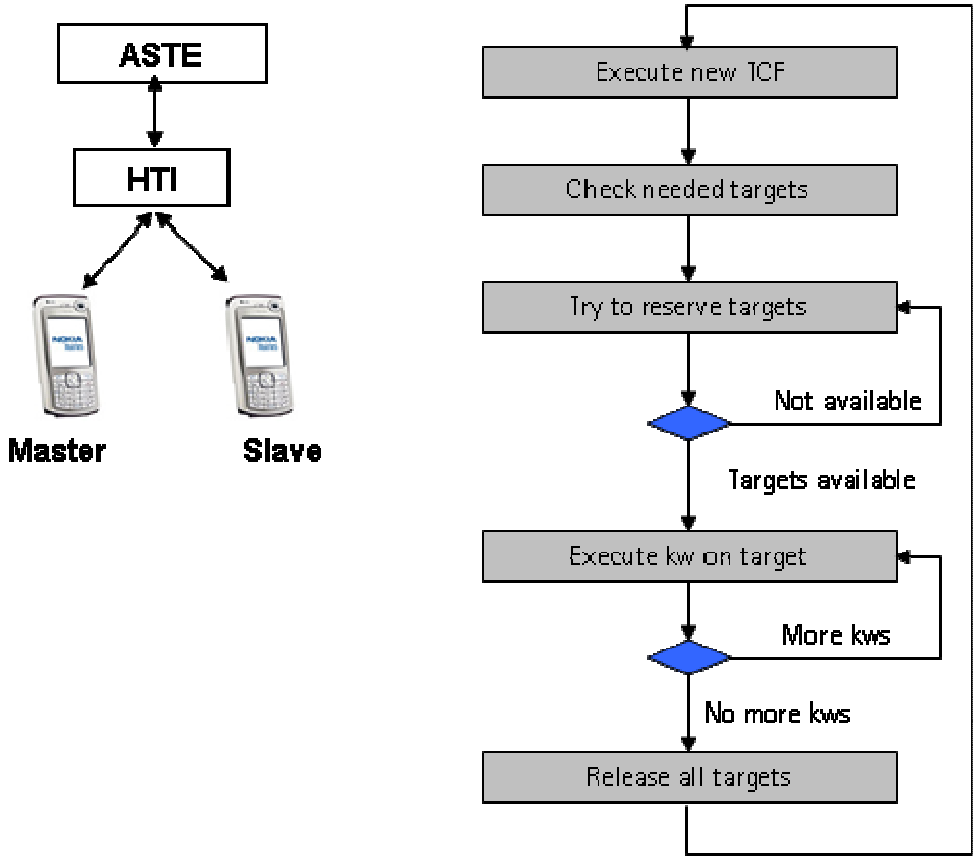
SUT:in (System Under Test) tietoihin on määriteltävä testaajan nimi, testilaite sekä toistokerrat ennen kuin testitiedoston suorittaminen voidaan aloittaa.

Testitiedostossa (TCF) on kerrottu testattavat testit. Siinä luetellaan muun muassa testin ID, avainsanat ja parametrit. Jos avainsana on linkitetty bittikarttaan, niin silloin käytetään BDF-tiedostoa linkittämään parametri oikeaan bittikarttaan näytöllä. Jos avainsana on linkitetty lokalisointiarvoon, niin käytetään lokalisointitiedostoa löytämään tarvittava teksti. Yhdessä testitiedostossa voi olla kaikki sovelluksen testit, eli useita satoja testejä. [13.]

Taulukko 3. Esimerkki testistä TCF- tiedostossa.

060a Add To-Do entry (#B #00000060)				
Test objectives:				
The purpose of the test case is to verify that the new To-do entry can be added				
Precondition(s):				
Device is in Todo application	kw_AppState	§Todo§		
Step 1:				
Open new entry dialog from Options menu	kw_PressHardkey	<SoftLeft>		
	kw_SelectMenu	§Add todo§		
Expected result:				
New entry dialog is opened				
Softkeys are				
§Options§			kw_VerifyText	§Options§
§Done§			kw_VerifyText	§Done§
Step 2:				
Set entry name to §TestEntry§	kw_Type	§TestEntry§		
Press right softkey	kw_PressHardkey	<SoftRight>		
Expected result:				
New entry is saved and the name §TestEntry§ is displayed in the list			kw_VerifyText	§TestEntry§

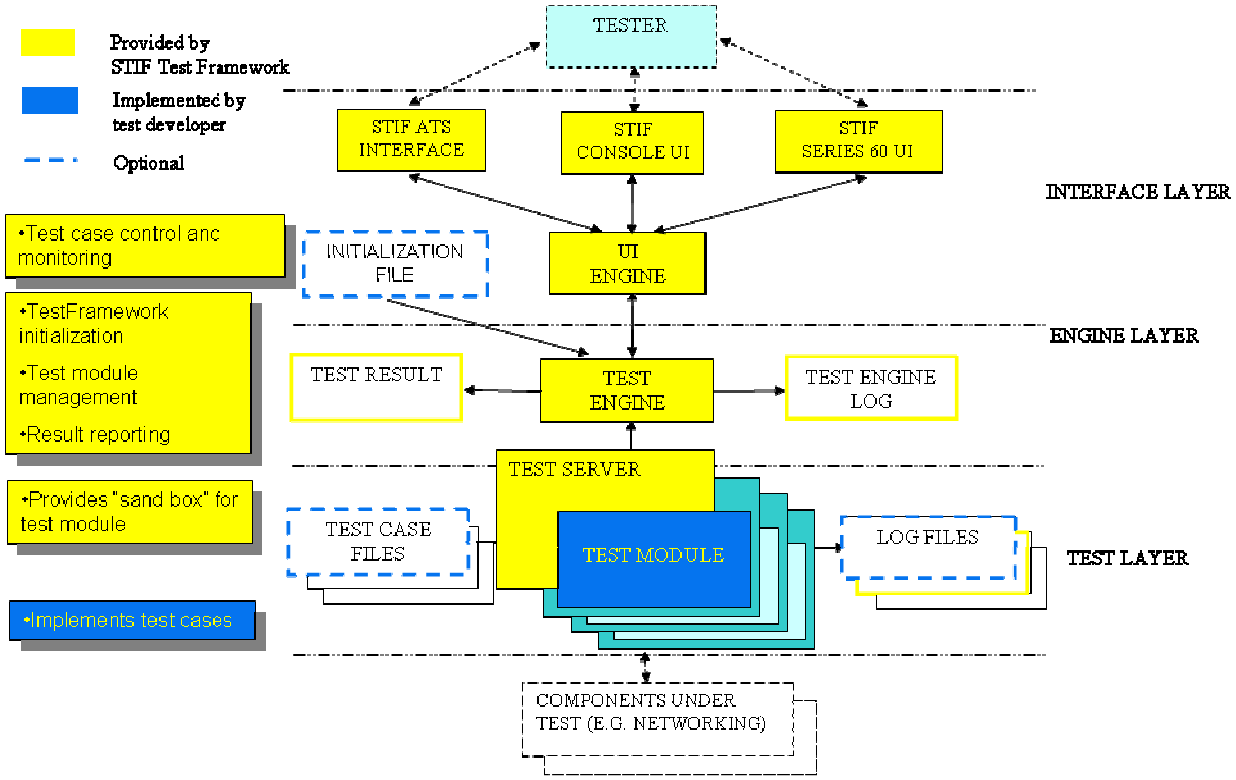
Master-Slave (isäntä-orja) ominaisuus ASTE:ssa mahdollistaa usean puhelimen hallinnan samassa testitiedostossa. Tämän ominaisuuden avulla onnistuu esimerkiksi tekstiviestin lähetys sekä puhelut puhelimesta toiseen. Master ja slave voivat olla eri puhelinmalleja, tai käyttää eri ohjelmistoversiota ja kieltä. Tämän avulla saadaan testaus lähemmäksi todellista tilannetta, ja mahdollisesti löydetään enemmän virheitä ennen kuin tuote toimitetaan asiakkaalle. [13.]



Kuva 11. MASTER-SLAVE- rajapinta sekä kohteen käsittely ASTE:ssa. [13.]

Kohteen (Target) käsittely ASTE:ssa alkaa testitiedoston suorituksella ja samalla tarkistetaan tarvittavat testilaitteet (kuva 11). Jos tarvittavia testilaitteita ei ole heti käytössä, jatkaa järjestelmä niiden etsimistä, kunnes löytää. Tämän jälkeen suoritetaan avainsanat. Lopulta kun kaikki avainsanat eli testit on suoritettu, siirrytään käsittelemään seuraavaa testitiedostoa.

ATS2.x- sekä ATS3-testausjärjestelmät käyttävät STIF:ia testauksen apuna. STIF on eräänlainen testausrajapinta Symbianin ei-käyttöliittymäkomponenteille. STIF Test Framework on työkalu testien implementoinnille sekä testaukselle. STIF käyttää erilaisia rajapintoja testauksessa: STIF Console UI:ta, STIF Series 60 UI:ta sekä STIF ATS Interfacea.

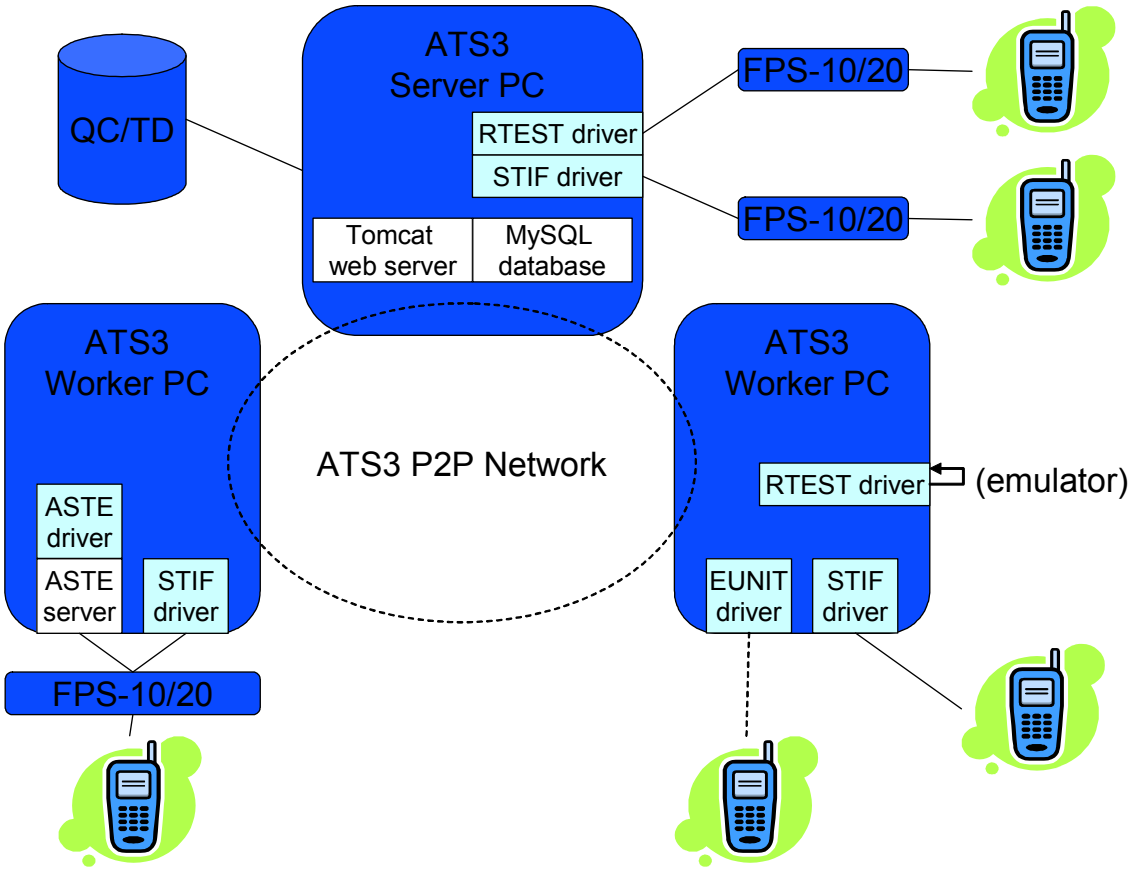


Kuva 13. STIF-arkkitehtuuri. [14.]

STIF:n ominaisuuksia on mm. virheiden ja poikkeuksien käsittely, muistivuotojen havaitseminen, ”Master-Slave” testaus käyttäen ATS- ja STIF-testejä sekä useiden testien suorittaminen yhtä aikaa. STIF asennetaan puhelimeen SIS-tiedostona. Puhelimeen asennetaan myös Testframework.ini-tiedosto, missä kerrotaan mitä moduulia halutaan testata. Tämän jälkeen avataan STIF-sovellus ja käynnistetään halutut testit.

ATS3

ATS3:ssa on yhdistetty ATS2.x:n ja ASTE:n ominaisuudet samaan pakettiin. Järjestelmässä on kaksi minimivaatimusta: ATS-tietokanta sekä ATS Server. Tarvittaessa ne voidaan asentaa samalle koneelle eli tarvitaan siis yksi PC, mutta suositeltavaa olisi erottaa tietokanta ja palvelin.



Kuva 14. ATS3-järjestelmän rakenne. [15.]

ATS3-järjestelmässä voidaan kaikki muut paitsi ASTE testitulokset siirtää Quality Centeriin. Järjestelmä sallii myös useiden ajurien rekisteröinnin yksittäiselle testilaitteelle, mutta jokaisella ajurilla pitää olla määriteltynä vain yksi testilaite (kuva 14).

Ennen kuin varsinainen ATS3 voidaan asentaa, täytyy koneelle asentaa Java SDK, MySQL ja Apache Tomcat. Kun nämä ohjelmistot on asennettu, voidaan asentaa varsinainen testausjärjestelmä. Asennuksen yhteydessä määritellään mm. käytettävät TCP-portit, tietokanta, käyttäjätunnukset ja salasanat. [16.]

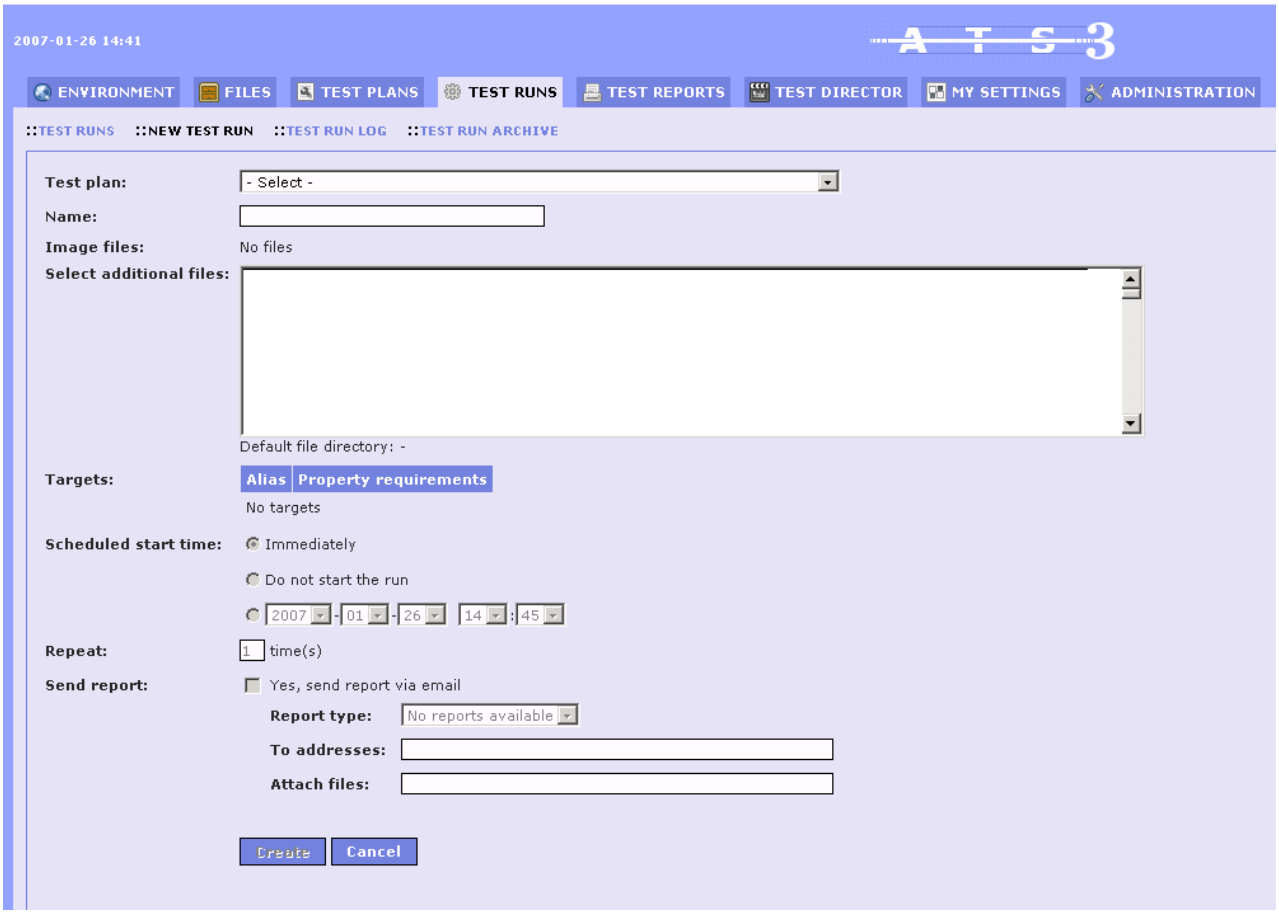
Testaus aloitetaan rekisteröimällä testattava laite haluttuun clientiin. Rekisteröinti tapahtuu määrittelemällä rekisteröintitiedostoon mm. testattavan laitteen nimi, kategoria, tyyppi ja yhteystapa. Jos testit suoritetaan matkapuhelimessa, pitää aluksi puhelimen

muisti tyhjentää, ja sen jälkeen asentaa tarvittava image. Image sisältää puhelimen perusohjelmiston. Tämän jälkeen puhelimeen asennetaan testattava ohjelmisto, STIF sekä HTI. Useimmiten puhelimen muistikortille asennetaan tarvittava testidata, esimerkiksi audiotiedostot, bittikartat jne. Tämän jälkeen puhelin kytketään halutulla tavalla ATS-järjestelmään.

```
NAME=Test phone1
CATEGORY=hardware
TYPE=Test phone1
CONNECTION=HTI
CLASS=StifTestableDevice
HARNESS=STIF
# reinstall files after reboot
REINSTALL=false
BUILD=urel
PLATFORM=armv5
#FLASHER=FPS10Tool
#single (.img) multi (.img.mcu && .img.ape) or optional_multi (.img.mcu (if exists) && .img.ape)
IMAGE_TYPE=optional_multi
#FPS10Tool or HTI
#REBOOTER=FPS10Tool
BOOT_TIME=5
#DataGateway port
DGW_PORT=
#Initialisation of DataGateway connection (note: only one should be uncommented)
#USB (DKU-2) Connection
#DGW_INIT_STRING=BUS_NAME: USB
#Standard FBUS Connection DAU-9(x)
DGW_INIT_STRING=BUS_NAME: FBUS PORT_NUM:1
#ComBox FBUS connection
#DGW_INIT_STRING=BUS_NAME: COMBOX PORT_NUM:1 COMBOX_DEF_MEDIA: FBUS
#ComBox Direct FBUS connection
#DGW_INIT_STRING=BUS_NAME: COMBOX PORT_NUM:1 COMBOX_DEF_MEDIA: FBUS BOX_TYPE: NEW
PROTOCOL: FBUS
#FPS-10 TCP-IP Connection ACTIVE_MEDIA set to FBUS, could be USB also
#DGW_INIT_STRING=
#FPS-10 USB Connection, ACTIVE_MEDIA set to FBUS, could be USB also
#DGW_INIT_STRING=
#FBUS Connection through USB to Serial converter DKU-5, CA-42, DKU-8
#DGW_INIT_STRING= BUS_NAME: DKU-5_FBUS PORT_NUM:80
#IrDA connection
#DGW_INIT_STRING=BUS_NAME: IRDA
```

Kuva 15. Esimerkki testilaitteen rekisteröintitiedostosta.

Rekisteröinnin jälkeen luodaan testisuunnitelma. Testisuunnitelmaan määritellään tarvittava testidata, testilaite sekä testausrajapinta. Testisuunnitelman jälkeen luodaan testiajo. Testiajoon valitaan haluttu testisuunnitelma. Valittua testisuunnitelmaa ei voi muokata testiajon luonnin jälkeen. Testiajoon määritellään myös testilaite, missä testit suoritetaan sekä ajankohta ja kuinka monta kertaa testit ajetaan. [16.]



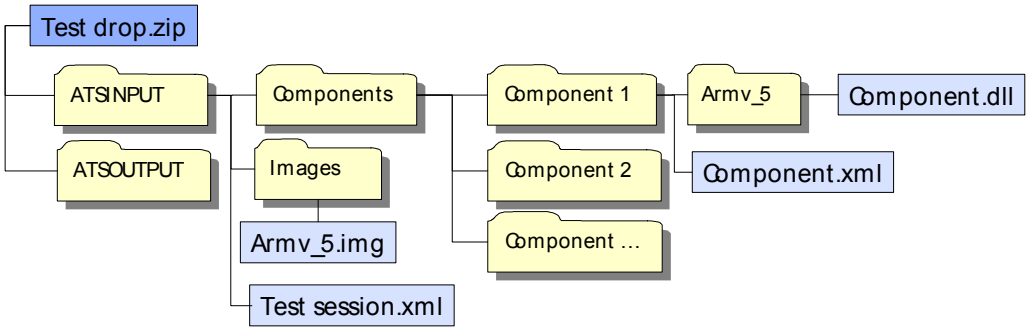
Kuva 16. Näkymä testiajon luonnista.

Testiajon suoritusta voidaan seurata reaaliaikaisesti. Testiraportti-osiosta näkee, missä vaiheessa testaus on sekä testitulokset. Testituloksissa kerrotaan mm. testien kokonaismäärä, monta testiä suoritettiin ja montako testiä meni läpi. Tuloksissa mainitaan myös testauksen kesto, ja eritellään, missä testissä virhe on ilmennyt.

6 EVALUOINTI

Työn tavoitteena oli selvittää, onko ATS3-järjestelmä tarpeeksi vakaa, testitulokset luotettavia ja voidaanko esimerkiksi siirtyä käyttämään ATS3:sta myös UI-testauksessa. Toistaiseksi testaukseen on käytetty ATS2.x-järjestelmää ja mm. grafiikkatestit on täytynyt suorittaa manuaalisesti.

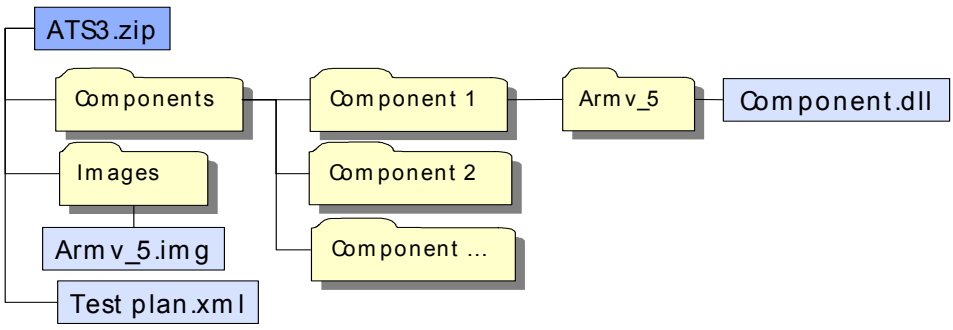
Testaus ATS3-järjestelmässä on erilaista verrattuna ATS2.x-järjestelmään. ATS2.x-järjestelmässä luodaan test drop.zip -tiedosto, joka ladataan palvelimelle. Tämä tiedosto sisältää kaiken informaation, jota tarvitaan testauksessa.



Kuva 17. Esimerkki millainen ATS2.x test dropin rakenne voi olla.

Test dropin tärkeimmät tiedostot ovat test session.xml, component.xml sekä component.dll. Test session.xml –tiedostossa luetellaan mm. kaikki testattavat komponentit sekä millä komponentti testataan eli testataanko se puhelimella vai emulaattorilla. Jokaisella komponentilla on oma component.xml-tiedosto, jossa määritetään, mitkä tiedostot asennetaan testilaitteeseen, sekä siinä voidaan määritellä, jätetäänkö tietyt testit suorittamatta. Component.dll-tiedosto sisältää ajettavaa ohjelmakoodia.

ATS3-järjestelmään voidaan ladata suoraan ATS2.x test drop. zip –tiedosto, jonka järjestelmä muokkaa sopivaksi, mutta tehokkainta on käyttää ATS3:n test planeja (testisuunnitelma).



Kuva 18. Esimerkki millainen ATS3:n test planin rakenne voi olla.

ATS3:n test plan.xml -tiedosto on periaatteessa component.xml-tiedoston sekä test session.xml -tiedoston yhdistelmä. Test plan -tiedostossa luetellaan mm. komponentit, testilaite ja testilaitteeseen asennettavat tiedostot.

Evaluoinnissa oli tarkoitus ajaa samat testit emulaattorissa ja matkapuhelimessa sekä ATS3-järjestelmän että ATS2.25-järjestelmän avulla ja verrata ovatko tulokset yhdenmukaiset. ATS3-järjestelmä ei ole vielä virallisesti käytössä, vaan se on tuotekehitysvaiheessa. Siitä havaittiinkin ohjelmistovirhe ensimmäisen testiajon jälkeen, joten ensimmäiset tulokset olivat virheellisiä.

Ohjelmistovirhe johtui siitä, että erääseen kenttään sijoitettava merkkimäärä oli rajoitettu 255 merkkiin. Tämä aiheutti sen, ettei *excluded*-ominaisuus toiminut oikein. *Excluded*–ominaisuudella voidaan määritellä, mitkä testit jätetään testaamatta kyseisestä komponentista. Järjestelmä testasi komponentin kaikki testit, eikä siis jättänyt testaamatta component.xml-tiedostossa *excluded*-tilaan määriteltyjä testejä.

Testaus onnistui hyvin testattaessa ohjelmistoa matkapuhelimessa, mutta emulaattorissa testaaminen ei onnistunut. Järjestelmä ilmoitti, että testejä ei voida ajaa HTI-yhteyden puuttumisen takia. Ongelmaa yritettiin selvittää myös testausjärjestelmän kehittäjän kanssa, mutta rajallisen ajan takia ei ongelmaa onnistuttu ratkaisemaan, joten testit ajettiin ainoastaan matkapuhelimessa.

Testitulokset

Evaluoinnissa ajettiin yhteensä viisi erilaista ajoa. Testiajo suoritettiin ensin ATS2.25-järjestelmässä ja toistettiin ATS3-järjestelmässä. Testien kokonaismäärien erot johtuvat siitä, että testiajoissa testattiin kahta eri ohjelmistoa ja niiden eri viikkojen versioita.

Taulukko 4. Ensimmäisen testiajon tulokset testattaessa ohjelmistoa matkapuhelimessa.

	ATS2.25	**ATS3**
Total	1270	1270
Executed	1270	1270
Passed	1254	1251
Failed	16	19
Pass rate	98.7%	98.5%
Run rate	100%	100%
Duration	01:54:25	01:24:42

Ensimmäisessä testiajossa tulokset olivat lähes yhdenmukaiset [taulukko 4]. Suoritettujen testien määrä oli sama, mutta ATS3-järjestelmässä läpi meni 1251 testiä, kun taas ATS2.25-järjestelmässä oli mennyt läpi 3 testiä enemmän. ATS3-järjestelmässä testien suorittaminen tapahtui n. 26 prosenttia nopeammin, mikä tarkoitti käytännössä 20 minuuttia.

Taulukko 5. Toisen testiajon tulokset testattaessa ohjelmistoa matkapuhelimessa.

	ATS2.25	**ATS3**
Total	3224	3223
Executed	3224	3223
Passed	3221	3221
Failed	3	2
Pass rate	99.9%	99.9%
Run rate	100%	100%
Duration	01:58:54	01:07:55

Toisessa testiajossa tulokset olivat myös lähes yhdenmukaiset [taulukko 5]. Suoritettujen testien määrä erosi ainoastaan yhdellä testillä sekä ei-läpäisseissä testeissä oli yhden testin ero. Testien suoritus tapahtui ATS3-järjestelmässä n. 43 prosenttia nopeammin, mikä käytännössä tarkoitti 51 minuuttia.

Taulukko 6. Kolmas testiajo.

	ATS2.25	**ATS3**
Total	2969	2968
Executed	2969	2968
Passed	2962	2965
Failed	7	3
Pass rate	99.7%	99.9%
Run rate	100%	100%
Duration	02:03:04	01:07:25

Kolmas testiajo sujui n. 45% nopeammin ATS3 järjestelmässä, mutta järjestelmä testasi yhden testin vähemmän [taulukko 6].

Taulukko 7. Neljäs testiajo.

	ATS2.25	**ATS3**
Total	1270	1212
Executed	1270	1212
Passed	1255	1198
Failed	15	14
Pass rate	98.8%	98.8%

Run rate	100%	100%
Duration	01:48:06	01:13:18

Neljännessä testiajossa testien kokonaismäärässä oli selkeä ero [taulukko 7]. Syy löytyi lopulta hakemistorakenteesta. Eli ATS2.x-järjestelmässä on käytössä Shared-kansio, joka sisältää laitespesifisiä tiedostoja, joita useampi komponentti voi käyttää (ATSINPUT/Components/Shared). ATS3-järjestelmän ohjelmistovirheen takia järjestelmä ei ymmärtänyt tuota rakennetta, joten se jätti testaamatta yhden komponentin tämän takia. Asia korjaantui sijoittamalla kyseinen tiedosto General–kansioon, joka sisälsi myös muita puhelimeen asennettavia tiedostoja (ATSINPUT/ Components/ ComponentX/ General). ComponentX_components.xml –tiedostoon tehtiin muutos vaihtamalla asennustyypiksi ”data” ”shared_binary”:n sijaan. Tämän jälkeen testien kokonaismäärä vastasi ATS2.25-järjestelmässä suoritettuun ajoon [taulukko 8].

Taulukko 8. Viides testiajo.

	ATS3
Total	1270
Executed	1270
Passed	1256
Failed	14
Pass rate	98.9%
Run rate	100%
Duration	01:24:00

7 ANALYSOINTI

Työn tavoitteena oli evaluoida voidaanko siirtyä käyttämään ATS3-järjestelmää. Kyseinen tuote on vielä tuotekehitysvaiheessa, joka tuli ilmi selkeästi evaluoinnin aikana. Järjestelmästä löytyi kaksi melko merkittävää ohjelmistovirhettä sekä emulaattoritestaus ei onnistunut lainkaan. Löytyneet ohjelmistovirheet raportoitiin ohjelmiston kehittäjälle, ja korjaukset luvattiin seuraavaan versioon. Matkapuhelintestauksen avulla paljastui uudesta järjestelmä monta hyvää ominaisuutta, ja tältä osin järjestelmä alkaakin olla hyvässä kunnossa.

Emulaattoritestauksen epäonnistuminen johtui todennäköisesti konfigurointivirheestä. Eli asetuksissa on määritelty väärin yhteystyyppi, portti tms. Mahdollista on myös, että ATS2.x-järjestelmän luomat test dropit eivät sovi suoraan ATS3-järjestelmään, vaan niitä pitää muokata. Jotta emulaattoritestaus saadaan toimimaan, tarvitaan sitä varten parempia ohjeita ohjelmiston kehittäjältä. Tältä osin ohjeet olivat hieman puutteelliset.

Matkapuhelintestaus onnistui uudessa järjestelmässä hyvin. Tulokset olivat lähes yhdenmukaisia, ja testauksen kokonaisaika nopeutui merkittävästi käytettäessä ATS3:sta. Testituloksista havaittiin, että testien kokonaismäärät erosivat hieman, sekä läpimenneiden ja epäonnistuneiden testien lukumäärät vaihtelivat. Tämä tarkoittaa käytännössä sitä, että vaikka testi X epäonnistui ATS2-järjestelmässä, se meni läpi ATS3-järjestelmässä. Syy voi johtua testattavan ohjelmiston epävakaudesta, mutta eroavaisuudet on selvitettävä ennen kuin järjestelmä otetaan käyttöön.

Testauksessa käytettiin suoraan ATS2.25-järjestelmää varten luotuja testidroppeja. ATS3-järjestelmä muutti test dropit tarvitsemaansa muotoon. Siirryttäessä testaamaan pelkästään ATS3-järjestelmällä, olisi järkevää ottaa käyttöön ATS3-järjestelmän mukaiset test planit. Tällöin testauksesta tulisi tehokkaampaa. Test planien käyttöönotto vaatii suuria muutoksia skripteihin, jotka luovat lähdekoodista tarvittavan testidatan. Muutoksia skripteihin tehdään todennäköisesti vasta sitten, kun voidaan ottaa ATS3-järjestelmä käyttöön, ja on koulutettu testaajat uuteen järjestelmään.

ATS3-järjestelmän uutta ominaisuutta ASTE:tta ei evaluoitu. ASTE:n evaluointi olisi ollut mielekästä, jos olisimme käyttäneet järjestelmää aiemmin UI-testaukseen. ASTE-ominaisuuden evaluointi vaatisi siis UI-testejä sekä tietysti tutustumista järjestelmään. Evaluointi tulee ajankohtaiseksi sitten, kun ATS3-järjestelmä on otettu käyttöön, ja aloitetaan UI-testauksen automatisointi.

ATS3-järjestelmän käyttöönotto viralliseksi testausjärjestelmäksi ei ole vielä ajankohtaista. Järjestelmää voidaan kuitenkin jo alkaa käyttämään rinnakkain ATS2.x-järjestelmän kanssa matkapuhelintestauksen osalta. Tämän avulla voidaan löytää järjestelmästä mahdollisia virheitä. Lopulta, kun virheitä ei enää löydy, voidaan siirtyä käyttämään pelkästään ATS3-järjestelmää. Tärkeää on myös tässä vaiheessa saada emulaattoritestaus kuntoon. Emulaattoritestaus on myös syytä suorittaa rinnakkain molemmissa järjestelmissä, jotta havaitaan mahdolliset eroavaisuudet. Tämän jälkeen voidaan keskittyä päivittämään olemassa olevat skriptit sellaisiksi, että voidaan ottaa test plan-tiedostot käyttöön. Kun ATS3-järjestelmä on lopulta käytössä, voidaan tarpeen tullen lopulta perehtyä ASTE-ominaisuuteen.

VIITELUETTELO

[1] Pyhäjärvi, Pöyhönen. Tehokas ohjelmistotestaus. [viitattu: 16.2.2007.] Saatavissa: http://www.cs.jyu.fi/~sakkinen/testaus/aineisto/1_TehokasOhjelmistotestaus_v1.1.ppt.

[2] Helsingin yliopisto, Juha Taina. Ohjelmistojentestaus- luentomateriaali.[viitattu: 16.2.2007.] Saatavissa: http://www.cs.helsinki.fi/u/taina/ohte/s-2004/luennot/Luku02.pdf.

[3] ISEB Foundation Certificate in Software Testing – materiaali.

[4] Luukkainen, Matti. Ohjelmistojen testaus. [viitattu: 16.2.2007.] Saatavissa: www.edu.**stadia**.fi/~luuma/testaus/materiaali/kalvot.ppt.

[5] Mäkelä, Sini. Testaustyökalut. Ohjelmistotuotantovälineet -seminaari. [viitattu: 16.2.2007.] Saatavissa: http://www.cs.helsinki.fi/u/laine/otv/testaustyokalut.pdf.

[6] Airaksinen, Kujanpää. Regressiotestaus osana testausprosessia. [viitattu: 16.2.2007.] Saatavissa: http://www.cc.jyu.fi/~aenevala/regressiotestaus.pdf.

[7] Ylikoski, Jukka. Lasilaatikkotestauksen kattavuuksien vertailu. [viitattu: 16.2.2007.] Saatavissa: http://www.cs.helsinki.fi/u/verkamo/sem/gradu_s2005/ylikoski2.pdf.

[8] Kautto, Tuomas. Ohjelmistotestaus ja siinä käytettävät työkalut. Ohjelmistotekniikan seminaariesitelmä. [viitattu: 16.2.2007.] Saatavissa: http://www.mit.jyu.fi/opiskelu/seminaarit/ohjelmistotekniikka/testaus/.

[9] Turun yliopisto, Ohjelmistotuotanto. IteratiivinenElinkaari. [viitattu: 16.2.2007.] Saatavissa: http://staff.cs.utu.fi/kurssit/ohjelmistotuotanto/06/IteratiivinenElinkaari.pdf.

[10] Huhtamäki, Mika. Oliopohjainen testaus. [viitattu: 16.2.2007.] Saatavissa: http://www.cs.helsinki.fi/u/mphuhtam/download/olio.pdf.

[11] http://www.testingstuff.com/tools.html. [viitattu: 16.2.2007.]

[12] Kantonen, Lauri. How to create test environment. [viitattu: 16.2.2007.] Yrityksen sisäinen dokumentti.

[13] Heimola, Antti. ASTE Introduction. [viitattu: 16.2.2007.] Yrityksen sisäinen dokumentti. ST_ASTE.ppt -esitys

[14] Heimola, Antti. Test tools & Automation introduction. [viitattu: 16.2.2007.] Yrityksen sisäinen dokumentti. ToolsIntro_June2006_0.1.ppt –esitys.

[15] Lappalainen, Pertti. ATS3 Quick overview. [viitattu: 16.2.2007.] Yrityksen sisäinen dokumentti. ATS3_Quick_overview.ppt –esitys.

[16] Heimola, Antti. InstallationGuide ATS3. [viitattu: 16.2.2007.] Yrityksen sisäinen dokumentti.