Arenduse tavad ja töökorraldus

Tellija jätab endale õiguse muuta raamlepingu alusel sõlmitavate hankelepingute tehnilises kirjelduses kirjeldatud nõudeid. Pakkuja peab lähtuma hankelepingute täitmisel järgnevatest nõuetest, v.a juhul kui tellija on hankelepingu tehnilises kirjelduses märkinud teisti.
Arendustööde teostamisel kohustub pakkuja lähtuma mittefunktsionaalsetest nõuetest.

1. Nõuded arenduste üleandmisele

Loodavad hanke tulemid antakse üle RIK-i koodirepositooriumi kaudu. Partneri meeskonnaliikmetele luuakse RIK-i poolt ligipääs koodirepositooriumile lähtekoodide, andmebaasimuudatuste ning muude tulemite või nende muudatuste üleandmiseks.

2. Commit

Arendamisel lisada commitide (lähtekoodi repositooriumisse lisamine) juurde kindlasti töökäsu nr, mille alusel muudatus sisse viiakse ning lühisõnaline lühikirjeldus, mida muudatused sisaldavad. Commit sõnumi alguses peab olema alati projekti nimi, mille raames commit tehakse ning kindlasti lisada info selle kohta, et kes antud commiti teeb (isiku nimi).
Vigade parandamisel tuleb alati committimisel lisada ticketi number märkusena.

Muudatusi tuleb commitida vähemalt kord päevas.

Enne commiti peab kood/teostatud töö/tulem olema:

1) Iseendaga kooskõlas (pole vastuolusid, katkiseid mooduleid, iseendaga vastuolus funktsionaalsust);
2) Üldharu/muu haru tulemiga kooskõlas (üldharus olemasolev pole peale commiti katki);
3) Edaspidi mugavalt kasutatav ja mõistetav, ehk sellel on olemas oma enda sisseehitatud testid, mis peale järgnevaid võimalikke *arendusi/commite/tegevusi* näitaks, kas see konkreetne funktsionaalsuse osa on endiselt korras, või läks katki ja tuleb korrastada.

Commititav *kood/teostatud töö/tulem* peab olema **piisavalt** testitud ja pakkuja poolt testidega kaetud, kasutades selleks:

a) *Unittest*'e,
b) *Systemtest*'e
c) Manuaalseid teste

3. Tagimine

Versiooni üleandmiseks tuleb sellest koostada tag. Tagi number peab ühtima versiooni numbriga, mida üle antakse, kuid tagi nimi peab alati algama projekti lühendiga. Tagi alla tuleb siduda kõikide arhitektuuriliste komponentide (rakendus, andmebaasiscriptid, x-tee adapter jne) ning muudetud või loodud dokumentatsiooni commitid.

Versiooni numbrid peavad olema vähemalt kolmekohalised (X.Y.Z, milles X- põhiversioon, Y - üleantav versioon, Z paranduse number). Vajadusel võib kokku leppida teistsuguses numeratsioonis.

http://tortoisesvn.net/docs/release/TortoiseSVN_en/tsvn-dug-branchtag.html#tsvn-dug-branch-1

Iga tagiga (uue versiooniga) peab kaasnema versiooni kirjeldus (mis taskid, mis ticketid, mis baasiscriptid, mis juhendeid täiendati jne), mis tuleb paigutada dokumentatsiooni alla.

4. Dokumentatsioon

Dokumentatsiooni tuleb hoida *trunk/doc/* kaustas. Alampuud lepitakse eraldi kokku.

5. Baasiscriptid

Andmebaas ning selle muudatused tuleb alati üle anda loovate, muutvate, täiendavate jne skriptidena ning neid tuleb alati hoida *trunk/baas/* kaustas.

Andmebaasiscriptide failinimed peavad olema kujul:

YYYYMMDDHHMM_prefiks+Järjekorranr.sql ehk 201105131430_KR0003.sql kus prefiks on süsteemi lühend.

6. Pakkuja poolne testimine

Partner peab lisaks kasutuslugude funktsionaalsele testimisele teostatava arenduse/töö/tulemi commitimisel ja üleandmisel ise tõestatavalt veendunud olema, et nende arendus/töö/tulem teeb seda, mis on nõutud ning samaaegselt ei riku/lõhu juba olemasolevat muud *arendust/tööd/tulemit*.
Iga kirjutatud koodiüksuse kohta peaks olemas olema ka vastav test, mis kindlustab, et loodu toimib ja seda ka eriolukordades. Samuti on testid abiks juba olemasolevate komponentide muutmisel, indikeerides koheselt võimaliku probleemolukorra.
Arendatav funktsionaalsus peab olema kaetud:

1. Unittestidega
2. Süsteemitestidega
3. Manuaalsete testidega

Igal commitimisel ning üleandmisega peab edukalt:

1. käivitama kõik Unittestid ning selle dokumenteerima.
2. käivitama kõik süsteemitestid ning selle dokumenteerima
3. käivitama kõik manuaalsed testid ning selle dokumenteerima
4. dokumenteerima, mis test data't kasutati, mis keskkondi kasutati, jne.

6.1 Unit-testid e. Moodultestid

Moodultestid testivad ÜHTE klassi isoleeritult teistest klassidest ja ressurssidest (andmebaas).
Ideaalis peaks igal iseseival klassil olema oma moodultest, mis kataks selle funktsionaalust täielikult.
Kuna klassid kasutavad teineteist, moodustuvad niiöelda kobarad, siis moodultesti ideoloogiaga see hästi kokku ei sobi. Selleks, et saaks testida tõesti vaid konkreetse ühe klassi funktsionaalsust, on kasutusel mock-objektid(väline komponent *Rhino Mocks*).

6.2 Süsteemitestid

Süsteemitestid testivad süsteemi kasutaja vaatenurgast, eesmärgiks on kontrollida süsteemi vastavust kasutaja nõudmistele.
Süsteemitestid jagunevad omakorda kaheks:

1. Funktsionaalsuse testid - üldjuhul koostatakse konkreetse teenuse klassi süsteemitestid sellises mahus, et kaetud oleksid selle kõik erinevad meetodid ning lisaks äriloogiliselt olulisemad stsenaariumid.
2. Mittefunktsionaalsete aspektide testid - näiteks jõudluse testimiseks.

6.3 Üldised nõuded testide kirjutamisel

Nagu eelnevalt sai kirjeldatud, on testid liigendatud oma olemuse järgi. Testi sisu ja iseloom varieerub olenevalt selle kategooriale (nt. [Test, Category("Bug")] – testis kirjeldatakse kindel veaolukord ning kontrollitakse selle põhjal vea olemasolu. testi andmed on antud veaolukorra järgi koostatud ja spetsiifilised). Küll aga on kujunenud kindlad head tavad, mida tuleb kõigi testide kirjutamisel järgida – koodieetikat, testide liigendamist, testide automatiseeritust (st. teste peab saama käivitada skriptidega projektiväliselt nt. kategooria alusel iga-öiselt) ning testide autonoomsust (testid peavad võimalikult hästi taluma ET muudatusi).

6.4 Koodieetika ja struktuur testide kirjutamisel

Testkood peaks olema liigendatud ja selgelt loetav:

1. Defineerida testis projekti erinevate osade kasutus (nt. *using System;*);
2. Iga testklass tuleb deklareerida *[TestFixture]*-raamistikus ning olenevalt testi iseloomust nimetada kujul Bug1234Test (kindlat viga #1234 kirjeldav Bug-test), Ticket1234Test (uuendust #1234 kontrolliv test) või TestiKirjutajaEesnimi_TeenuseNimiTest (teenuse spetsiifiline süvatest);
3. Testis olevad privaatsed klassimuutujad on tava kohaselt kujul *_muutujaNimi*, näiteks
   ```
   private Menetlus _menetlus;
   ```
4. Kaitstud ja avalikud muutujad on kujul muutujaNimi või MuutujaNimi;
5. Vajalik uute klassiobjektide initsialiseerimine (nt. liidesed) toimub meetodis *SetUp()*. Näiteks:
   ```
   [SetUp]
   public void SetUp()
   {
   _liides = new Liides();
   }
   ```
6. Iga uus testcase tuleb kirjutada raamistiku sisse kujul
   ```
   [Test, Category("Kategooria")]
   ```
   ning testi enda nimetus tuleneb testi tüübist:
   a) Bug-testide ja uuendusi kontrollivate testide korral nimetus, mis kirjeldab võimalikult hästi ja konkreetselt kontrollitavat objekti/olukorda .
   b) Teenusepõhiste süvatestide puhul esiteks test, mis kontrollib, et teenust saaks nii minimaalsete kui ka maksimaalsete andmetega kinnitada, nimetusega A_SetUp()
7. Testandmed võiksid olla võimalikult reaalse elu lähedased ja loogilised, st. ei tohiks tekitada uut objekti klassist Helikopter kujul *_a*, vaid kujul *_helikopter*.
8. Iga äridokumendis olev nõue või tingimus tuleb testida eraldi testcase'i all, olenemata sellest kas kontrolliks on üks või rohkem ridu koodi. Kui teha äriloogiline kontroll vaid ühe testi all, siis ei kajasta see teenuse tervist – st. kui esimene kontroll annab veateate, siis ülejäänuteni test ei jõua. Selle vältimiseks on testid ärinõuete kaupa eraldi kirjutatud.

6.5 Autonoomsus

Testid peavad taluma pisemaid koodi muudatusi (neid, millega ei kaasne DTO või andmemudeli muutused).

6.6 Automatiseeritus

Kõiki teste peab saama käivitada näiteks projektiväliselt skripti abil. Selleks peavad testid olema õigesti kategoriseeritud.