Lähtekoodi hindamiskriteeriumid

LÄHTEKOODI HINDAMISKRITEERIUMID

Rahandusministeeriumi infotehnoloogiakeskus

Arhitektuur

Kontseptuaalne terviklikkus

Kontseptuaalse terviklikkuse all tuleb hinnata eelkõige sarnaste probleemide lahendamist sarnaste lahendusviisidega, misläbi tekivad ühtsed arhitektuurimallid ja läbivad loogikamustrid.

Loogilised abstraktsioonitasemed, arhitektuurikihid ja modulaarsus

Keerulised süsteemid peavad olema üles ehitatud loogiliste kihtidena ja tükkidena, millest igaüks täidab oma konkreetset rolli ja eesmärki.

Põhimõte, mida eelkõige siinkohal silmas pidada on „ *Separation of Concerns* “.

Koodi struktureerimine

Üldine projektistruktuur ja koodipaketid peavad järgima ühtseid lähenemisviise ning toetama modulaarsuse ja loogiliste abstratsioonitasemete põhimõtteid.

Muuhuldas tuleb pakendamisel ja sõltuvuste haldamisel hoiduda ringahelate (*Circular Dependency*) ohtudest.

Integratsiooniliideste ülesehitus

Integratsiooniliidesed peavad olema eraldatud sisemisest domeeniloogikast ning järgima soovitavalt mõnda üldlevinud disainimustritest ning soovituslikult publitseerima SOAP või REST teenusliideseid.

Mõned levinud tüüpmustrid, mida vastavalt vajadustele rakendada on „ *Service Gateway“* , „ *Remote Facade“*, *„Proxy“*

Veahalduse ja logimise ülesehitus

Veahalduse ja logimise ülesehitus on väga oluline rakendus- ja süsteemitaseme halduse seisukohalt ning eelkõige tuleb jälgida vigade kontekstiloogika selgestimõistetavust.

Administraatoritele on oluline veatöötluse ja teavituste mugavad konfigureerimisvõimalused, samuti tuleb hinnata konteksti sobivate veakoodide ja erindite kasutamist ning vea haldusvoogude läbipaistvust.

Logimise ja teavituse mehhanismid on tihtipeale süsteemi läbivad ning seetõttu on soovituslik logimistega seonduv loogika sobivale tasemele abstraheerida ning kasutada võimalusel aspekt-orienteeritud lähenemisviise.

Skaleeruvus ja jõudlusorienteeritus

Süsteemide jätkusuutlik kasvamine ja arenemine on hõlbustatud läbi piisava süsteemse modulaarsuse ning korrashoiu tagamisega., mida toetavad sellised põhimõtted nagu „ *Refactoring“* , „ *Evolving Architecture & Design“* , *„Managing Technical Debt“*

Käitusaegse skaleeruvuse ja jõudlusorienteerituse seisukohalt tuleb lisaks päringute optimeerimisele pöörata tähelepanu vahepuhvrite rakendamisele ja süsteemi ressursitarbe efektiivsele ohjamisele.

Üks oluline arhitektuuriline põhimõte kõrgete käideldavusvajaduste toetamiseks on „ *Command/Query Responsibility Segregation Principle “*.

Turvalähenemise ülesehitus

Erinevaid turvalähenemise ülesehitusi on palju ning konkreetse süsteemi valikut ja teostust tuleb hinnata eelnevalt püstitatud konkreetsetest funktsionaalsetest ja mittefunktsionaalsetest nõuetest.

Toetavate tugiraamistike rakendamine

Juhul kui on süsteem on üles ehitatud tuginedes mõnele levinud raamistikule (Spring, Tapestry, Hibernate, Wicket, JSF jne), siis tuleb raamistiku rakendamist hinnata vastavalt selle raamistiku üldlevinud headest tavadest lähtvuvalt.

Levinud tavad ja asjakohased seletused on kirjeldatud konkreetsete raamistike arendusjuhendites ning kommuunides, mille alusel tuleb koodis rakendatud lähenemisi hinnata.

Tihtipeale on üheks oluliseks tugiraamistiku valikukriteeriumiks piisava arendusdokumentatsiooni, praktikate ja kommuuni olemasolu.

Disain ja koodistiil

Põhiküsimused

1. Kas disain on lihtsalt mõistetav ?
2. Kas disain järgib süsteemi arhitektuurikonteksti ?
3. Kas kood implementeerib disaini korrektselt ?

Levinud disainimustrite rakendamine

Suuremate ärisüsteemide arendamisel kasutatakse sageli levinud disainimustreid, mis tagavad lahenduste terviklikkuse ning headele praktikatele tuginevad tõestust leidnud lahenduskäigud.

Mõned olulised tüüpmustrid, mida jälgida:

- *Gang of Four* baasmustrid
- *Domain Driven Design* domeenimudeli muster
- *Model-View-Controller* veebikarkassi muster
- *Dependency Injection* sõltuvusjuhtimise muster

Koodi „isedokumenteeriv“ stiil

Hästimõistetavas koodis nimetatakse kõik paketid, klassid, meetodid, muutujad võimalikult konteksti peegeldavad ja tähenduslikud. Tähenduslikku ja modulaarset koodi on tulevikus palju lihtsam hallata ning jätkuarendada, mis tähendab tellijale väiksemaid kulutusi.

Dokumentatsioon ja kommentaaride kasutamine

Isedokumenteerivale koodile ei ole tavapäraselt vaja lisada massiliselt selgitavaid koodikommentaare, vaid mõistetavalt nimetatud koodi struktuurielemendid juba paljastavad piisavalt arenduse ärilist ja tehnilist tausta.

Samuti ei ole vajadust massiliselt kasutada JavaDoc’i dokumenteerimist ning pigem lisada seda süsteemiosadele, millest arusaamine on vajalik osapooltele, kes otseselt koodi ei sirvi või pole selleks ligipääsusid (näiteks integratsiooniarendajad, testijad, rakendusadministraatorid)

Nõuetekohane JavaDoc on soovituslik tagada eelkõige väliste integratsiooniliideste ja peamiste süsteemifunktsionaalsust piiritlevate sisemiste teenusliideste kirjeldamiseks, mis tagab piisava täiendava dokumentatsiooni süsteemi sisend- ja tugipunktide mõistmiseks.

Staatiliste koodianalüsaatorite kontrollid

Koodi ülevaatusel on soovituslik rakendada üldlevinud vahendeid, mis juhivad tähelepanu koodi ebakorrektsusele ja võimalikele probleemkohtadele. Minimaalselt tuleb tähelepanu pöörata järgmistele punktidele:

- IDE kompilaatorite ja validaatorite hoiatused (nt Eclipse, http://www.eclipse.org/)
- PMD vahendi kontrolltulemused, koos *Copy/Paste Detector* seadistusega (http://pmd.sourceforge.net)
- FindBugs vahendi kontrolltulemused (http://findbugs.sourceforge.net/)
- Checkstyle vahendi kontrolltulemused (http://checkstyle.sourceforge.net)

Ehitus ja paigaldus

Automatiseeritud ehitus

Tuleb hinnata ehitusskriptide ülesehitust ning erinevate sihtkeskkondade konfiguratsioonijuhtimise võimalusi.

Lisaks hinnata levinud ehitusvahendite nagu Ant ja Maven rakendamist ning nendega seotud häid praktikaid ja kasutusviise.

Paigaldus

Hinnata tuleb automatiseeritud ja manuaalsete tegevuste osakaalu ning lisaoptimeerimise võimalusi.

Samuti tuleb hinnata administraatorite tarbeks loodud juhendite lihtsust, korrektsust ja paikapidavust.

Continuous Integration

Juhul kui projektis on rakendatud CI lähenemist, siis tuleb hinnata sellest tulenevaid protsessilisi ja valitud platvormiga seonduvaid häid tavasid ja praktikaid.

Lisaks tuleb hinnata lähenemist kvaliteedijuhtimise seisukohalt ning võimalike defektide avastamise võimekust.

Testitavus

Ühiktestid (*Unit tests*)

Ühiktestide olemasolu on kindlasti plussiks, mis näitab muuhulgas ka arendajate tööstiili ja vastutust koodikvaliteedi osas. Aina rohkem edasijõudnud arendajaid loob koodi *Test Driven* metoodikate põhimõttel, misläbi luuakse suuremat väärtust tarkvara terviklikule elutsüklile.

Integratsioonitestid (*Integration Tests*)

Pragmaatilisest seisukohast lähtuvalt peaks olema automatiseeritud integratsioonitestidega minimaalselt olema kaetud süsteemi äriloogika peamised sisend- ja integratsioonipunktid.

Näiteks kui on süsteemis kasutatud *Service Facade* disainimustrit ning arhitektuuriliselt joonduvad välja selgepiirilised teenuskomponendid, siis on kindlasti otstarbekas luua teenuskihile põhiloogikat testivad integratsioonitestid.

Juhul kui lahenduse loomisel on kasutatud Spring raamistikku, siis on kindel soovitus rakendada ja hinnata selle testimisvõimalusi ja abistavaid API-sid.

Testimisraamistike kasutamine

Tõenäoliselt on kasutatud testide loomiseks ja käivitamiseks enamlevinud testraamistikke nagu JUnit või TestNG. Sellest lähtuvalt tuleb hinnata nende raamistike kasutamist ja seonduvaid praktikaid ja põhimõtteid.

Üldhinnang

Lõplik kokkuvõttev üldhinnang

Lõplikus kokkuvõttes tuleb anda üldine hinnang ülevaatuse all olnud süsteemile ja detailsel ülevaatusel leitud punktidele. Üldhinnang on suunatud pigem mitte süvatehnilistele isikutele ülevaatliku mulje saamiseks.

Puuduste hinnanguline mõju tellijale

Lõpetuseks tuleb hinnata erinevate leitud puuduste tõsidust ja mõju tellijale nii lühemas kui pikemas perspektiivis. Samuti on soovituslik leitud puudusi klassifitseerida ja prioritiseerida nende tõsidusest lähtuvalt.

Viited

Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin)

Code Complete, Second Edition (Steve McConnell)

Gang of Four patterns (http://en.wikipedia.org/wiki/Design_Patterns)

Patterns of Enterprise Application Architecture (http://martinfowler.com/eaaCatalog)

J2EE patterns (http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html)

EAI patterns (http://www.eaipatterns.com)

Continuous Integration (http://martinfowler.com/articles/continuousIntegration.html)