Suunnittelutaktiikoita

Luento 6
1. osa

Oppimistavoitteet

- Suunnittelutaktiikoita laatuominaisuuksien saavuttamiseksi
  - Suorituskyky-, muunneltavuus- ja testattavuustaktiikat

SUUNNITTELUTAKTIIKAT

Laatutekijät arkkitehtuurisuunnittelussa

- Onnistunut arkkitehtuuri on edellytys järjestelmän keskeisten laatuvaatimusten täyttymiseksi
  - Tyylien ja patternien soveltaminen suunnittelussa auttaa haluttujen laatuominaisuuksien saavuttamisessa
- Yksittäinen arkkitehtuurityyli tai patterni ei kuitenkaan ota kantaa *kaikkiin* mahdollisiin laatuvaatimuksiin
  - Patterneja täytyy sovittaa ja muokata suunnittelun kohteena olevan järjestelmän (SUD, system under design) spesifisten vaatimusten, rajoitteiden ja muun kontekstin perusteella
  - ”The devil is in the details”

Laatutekijät arkkitehtuurisuunnittelussa

- Laatuominaisuudet (a.k.a laatuattribuutit, laatutekijät) vaikuttavat toisiinsa
  - Monilla laatuominaisuuksilla on negatiivinen vaikutus suorituskykyyn, esim. ylläpidettävyyttä ja muokattavuutta parantavat modularisointi ja abstrahointi lisäävät olioiden ja niiden välisen kommunikaation määrää ja pidentävät kutsuketjuja
  - Laatuominaisuuksien tasapainottelua (trade-off) ei voi yleensä välttää (esim. Rackspace –tapaus 1. luennolta)
- Ei voida keskittyä laatuominaisuuksiin yksittäin, vaan kokonaisuus ratkaisee

Laatutekijät arkkitehtuurisuunnittelussa

- Laatuattribuutteja
  - accessibility, accountability, accuracy, adaptability, administrability, affordability, auditability , autonomy, availability, credibility, process capabilities, compatibility, composability , configurability, correctness, customizability, degradability, determinability, demonstrability, dependability, deployability , distributability, durability, efficiency, evolvability, extensibility, failure transparency, fault-tolerance, fidelity, flexibility, installability, Integrity, interchangeability, interoperability , learnability, maintainability , mobility, modularity,nomadicity, operability. orthogonality, portability, precision, predictability. provability. recoverability, relevance, reliability, repeatability, reproducibility, resilience, responsiveness, reusability, robustness, safety, scalability, seamlessness, self-sustainability, serviceability, securability, simplicity, stability, standards compliance, survivability, sustainability, tailorability, testability, timeliness, traceability, ubiquity, understandability, upgradability, usability

  (Wikipedia)
- Ohjelmistojen laatumalli: ISO 25000 standardisarja (a.k.a *SQuaRE*, luettavissa Kumpulan tiedekirjastossa)
  - Käsitellään “Ohjelmistoprosessit ja ohjelmistojen laatu” -kurssilla

Laatutekijät arkkitehtuurisuunnittelussa

- Bass et al.[1] ovat määritelleet *taktiikoita* laatuominaisuuksien saavuttamiseen
  - Taktiikka on *suunnitteluprimitiivi*, joka vaikuttaa tietyn laatuominaisuuden toteutumiseen
  - Tyylejä ja malleja yksityiskohtaisempia ja lähempänä toteutusta ja yleiskäyttöisiä
    - Voidaan soveltaa monessa eri kontekstissa
- Tarkastellaan esimerkkeinä *suorituskykyä, ylläpidettävyyttä* ja *testattavuutta*

[1] http://etutorials.org/Programming/Software+architecture+in+practice,+second+edition/Part+Two+Creating+an+Architecture/Chapter+5.+Achieving+Qualities/

SUORITUSKYKY

Suorituskyky

- *Suorituskykytaktiikoiden* tavoitteena on saavuttaa määrätty reaktio-/vastausaika (response time) johonkin järjestelmän vastaanottamaan tapahtumaan ja saavuttaa haluttu suoritusteho (throughput)
- *Vastausaika* ja *suoritusteho* riippuvat
  - 1. resurssien käytöstä (esim. kuinka paljon tarvitaan datan luku-/kirjoitusoperaatioita) ja resurssien omnaissuorituskyvystä (kauanko kestää lukea/kirjoittaa 1Mb dataa levylle)
  - 2. odotusajasta
    - Kilpailu resursseista
    - resurssin saatavuus
    - Riippuvuus muusta toiminnasta
- Resurssi*tarpeeseen* vaikuttavat taktiikat (= kuinka paljon prosessointia saapuvan työn tekeminen vaatii)
  - laskennan tehostaminen (algoritmi, välitulosten käsittely)
  - yleisrasitteen vähentäminen (esim. etäkutsut paikallisiksi)
  - tapahtumamäärän vähentäminen
  - tapahtumien vastaanoton kontrollointi
  - käsittelyajan rajoittaminen
  - jonojen koon rajoittaminen
  - Huom: Laskennan määrää vähennettäessä joudutaan usein tinkimään tulosten tarkkuudesta

Suorituskyky

- Resurssien *hallinta*taktiikat (= kuinka paljon ja millaisia resursseja käytetään)
  - Rinnakkaisuuden lisääminen
  - Toiminnan / datan monistaminen
  - Resurssien/tehokkaampien resurssien lisääminen
- Resurssien *allokointi*taktiikat (= miten eri työt saavat resursseja käyttöön)
  - jono
  - prioriteettijono
  - dynaaminen prioriteettijono
  - staattinen allokointi

[Ohjelmistojen suorituskyvyn suunnittelusta on oma kurssinsa]

MUUNNELTAVUUS

Muunneltavuus

- Muunneltavuustaktiikat (modifiability tactics)
  - Ohjelmiston muunneltavuutta mitataan laskemalla aikaa ja kustannuksia, jotka kuluvat ohjelmistomuutosten toteuttamiseen, testaamiseen ja käyttöönottoon
    - Korjaukset, uudet toiminnot, toimintaympäristön muutokset jne.
  - Muunneltavuutta lähellä olevia tekijöitä: adaptability, evolvability, maintainability
- Muunneltavuustaktiikat voidaan jakaa kolmeen ryhmään
  - Muutosten lokalisointiin perustuvat taktiikat
  - Heijastusvaikutusten (ripple effect) estämiseen perustuvat taktiikat
  - Sidonnan viivästämiseen perustuvat taktiikat

Muunneltavuustaktiikat Muutosten lokalisointi

- Tavoitteena *rajoittaa* odotettavissa olevien *muutosten vaikutus* mahdollisimman pieneen joukkoon komponentteja
  - Määrätään komponenttien vastuut siten, että *odotettavissa olevat muutokset* koskevat rajattua joukkoa komponentteja
- Lokalisointitaktiikkoja:
  - Komponentin vastuiden semanttinen yhtenäisyys (semantic coherence)
    - Vastuut hoidettavissa ilman laajaa ulkopuolista kommunikointia
  - Yleiskäyttöisten palvelujen eristäminen omiin moduuleihin
  - Odotettavissa oleviin muutoksiin varautuminen
  - Yleistäminen ja parametrisointi
    - Erikoistaminen parametrien kautta – parametrien tulkinta
  - Vaihtoehtojen rajoittaminen
    - Korvattavien komponenttien/palvelujen tyypin rajoittaminen

Muunneltavuustaktiikat Heijastusvaikutusten esto

- Ohjelmamuutoksen heijastusvaikutukset (*ripple effects*) tarkoittavat muutoksia, jotka täytyy tehdä moduuleihin*, joita varsinainen muutos ei suoraan koske
- Jos moduulia A muutetaan, niin moduulia B joudutaan muuttamaan vain siksi, että A:ta on muutettu – ei siksi että B:n ominaisuuksia on ollut tarve muuttaa
  - Tämä heijastusvaikutus johtuu siitä, että B:n toteutuksessa on jokin riippuvuus A:n toteutukseen

*) voitaisiin useimmissa tapauksissa puhua myös komponenteista

Muunneltavuustaktiikat Heijastusvaikutusten esto

- Moduulien välisiä riippuvuustyyppejä:
  - Syntaktinen (muotoa koskeva) riippuvuus koskien dataa tai palveluja
    - Yhteinen formaatti datalle, palveluiden kutsumuodot
  - Semanttinen (merkitystä koskeva) riippuvuus koskien dataa tai palveluja
    - Datan/palvelun käyttäjän on tulkittava merkitys samoin kuin tuottajan
  - Järjestysriippuvuus koskien dataa tai kontrollia
    - Datan osien saapuminen tietyssä järjestyksessä
  - Palveluiden suorittaminen tietyssä järjestyksessä
    - A.n pitää olla suorittanut palvelu A1 ennen kuin B voi suorittaa palvelun B1
  - Rajapinnan identiteetti tunnettava
    - Nimi tai kahva ("handle") oltava tiedossa käännös ja suoritusaikana

Muunneltavuustaktiikat Heijastusvaikutusten esto

- Moduulien välisiä riippuvuustyyppejä…
  - Suoritusaikainen sijainti tunnettava
    - Esim. IP-osoitteen tietäminen
  - Datan tai palvelun laatutaso tunnettava
    - esim. tarkkuus (onko sentit mukana?)
  - Komponentin olemassaolon edellyttäminen
    - Ei voida luoda dynaamisesti
  - Moduulin käyttämien resurssien tunteminen

Muunneltavuustaktiikat Heijastusvaikutusten esto

- Semanttisiin riippuvuuksiin perustuvia heijastusvaikutuksia on yleisesti ottaen vaikea estää
- Voi olla epätarkoituksenmukaista tarjota kaikille käyttäjille semantiikaltaan täsmälleen samanlaista rajapintaa ja käyttäytymistä kuin ennen muutoksia
  - Vanhan implementaation muuttamiseen on yleensä painavia syitä
  - Kahta erillistä implementaatiota voidaan joutua ylläpitämään siirtymäkauden ajan (vanhaa uuden rinnalla)
- Seuraavassa esitetyistä taktiikoista mikään ei välttämättä estä semanttisiin riippuvuuksiin perustuvia heijastusvaikutuksia

Muunneltavuustaktiikat Heijastusvaikutusten esto

- Tiedon piilotus (Information hiding)
  - Odotettavissa oleviin muutoksiin varautumista
    - Todennäköisesti muuttuvien ominaisuuksien piilottaminen on moduulijaon yhtenä pääperiaatteena
- Olemassa olevien rajapintojen säilytys (Maintain existing interfaces)
  - Jos B riippuu A:n tarjoaman rajapinnan nimestä tai kutsumuodoista, rajapinnan ja syntaksin säilyttäminen tarkoittaa, ettei B:tä tarvitse muuttaa A:n päivityksen yhteydessä
    - Ei välttämättä auta, jos riippuvuudet ovat semanttisia tai palveluiden/datan laatutasoon tai resurssien käyttöön liittyviä
    - Rajapintojen lisääminen, sovittimen (adapter) tai edustajaolion käyttö (proxy)

Muunneltavuustaktiikat Heijastusvaikutusten esto

- Kommunikaatioväylien rajoittaminen
  - Rajoitetaan niiden moduulien määrää, joiden kanssa tietty moduuli käyttää samaa dataa
  - Rajoitetaan niiden moduulien määrää, jotka tuottavat tietyn moduulin käyttämää dataa tai jotka käyttävät sen tuottamaa dataa
- Välittäjän käyttö
  - Jos B:n riippuvuus A:sta ei ole semanttista tyyppiä, on aina mahdollista lisätä välittäjä B:n ja A:n väliin
  - Välittäjä piilottaa riippuvuuden konkreettisen ilmentymän ja tarjoaa B:lle muuttumattoman näkymän A:han
  - Välittäjä vastuulla on vain riippuvuuden hallinta, ei mitään muuta
  - Erilaisia välittäjiä eri tarpeisiin: nimipalvelin, resurssimanageri

Muunneltavuustaktiikat Sidonnan viivästäminen

- Edellä kuvatut taktiikat pyrkivät vähentämään muutosten kustannuksia pienentämällä muutettavien moduulien määrää
- Niistä ei kuitenkaan ole hyötyä, jos halutaan lisätä ohjelmiston muunneltavuutta koskien moduulien/komponenttien käyttöönottotapaa ja –aikaa tai halutaan mahdollistaa muiden kuin kehittäjien tekemän muutokset

Muunneltavuustaktiikat Sidonnan viivästäminen

- *Sidonnan viivästäminen* tukee molempia tavoitteita
  - Periaate: lisäominaisuus tai muunneltu toiminnallisuus voidaan ottaa käyttöön ja kytkeä muuhun ohjelmistoon *ajoaikana* (vs. koostamisaika, build-time)
- Nopeuttaa käyttöönottoa: järjestelmää ei tarvitse ajaa alas (ja käynnistää uudelleen)
- Vaatii lisäinfrastruktuuria -> kasvattaa kustannuksia

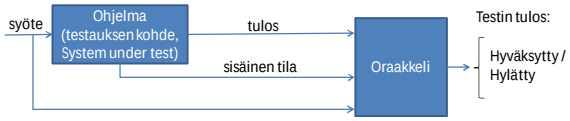
Muunneltavuustaktiikat Sidonnan viivästäminen

- Ajonaikainen rekisteröinti (runtime registration) mahdollistaa plug-and-play operaatiot. Rekisteröinti aiheuttaa lisäkustannuksia. Tuottaja/kuluttaja rekisteröinti voidaan tehdä ajoaikaisesti tai latauksen yhteydessä.
- Konfigurointitiedostoissa voidaan säädellä käynnistyksen yhteydessä tehtäviä sidontoja ja parametreja.
- Polymorfismi mahdollistaa metodikutsujen myöhäisen sidonnan.
- Komponentin korvaus mahdollistaa latausaikaisen sidonnan.
- Protokollien noudattaminen mahdollistaa prosessien ajonaikaisen sidonnan
- Vrt. *Microservices* –tyylissä toimintojen jako ja deployment

TESTATTAVUUS

Testattavuustaktiikat

- Ohjelman tai sen osan (komponentin) testauksen yleinen malli:



Testattavuustaktiikat

- Jotta ohjelmistoa (tai sen osaa) voi mielekkäästi testata, on voitava
  - Kontrolloida ohjelman ja sen osien syötteitä
  - Havaita sen tuottamat tulokset
  - Ja joissain tilanteissa myös voitava tarkkailla ohjelman sisäistä tilaa syötteen käsittelyn aikana tai välittömästi käsittelyn jälkeen
- Tähän tarvitaan yleensä varta vasten rakennettu *testipeti* (test harness)
  - Testien laatimisen ja suorittamisen sekä testipetien rakentamisen kustannukset voivat olla huomattavat
  - Testien suorituksen ja tulosten raportoinnin automatisointi aivan olennaista ketterässä kehitysessä (TDD, CI)

Testattavuustaktiikat

- Syötteiden ja tulosteiden kontrollointi
  - Record/Playback: Komponenttien rajapintojen kautta tulevien syötteiden nauhoittaminen ohjelman suorituksen aikana, ja nauhotettujen syötteiden käyttö testipedissä testitapausten syötteinä
  - Komponenttien/palvelujen rajapinnan erottaminen niiden toteutuksesta: palvelut, joista testin kohteen suoritus riippuu, voidaan korvata tyngillä tai virtualisoida (stub, *mocking*), jotta komponentteja voidaan testata erillään ja saada aikaan täsmälleen haluttuja tilanteita (esim. harvinaisia poikkeuksia)

Testattavuustaktiikat

- Syötteiden ja tulosteiden kontrollointi (jatkuu)
  - Erikoisrajapinnat testausta varten: Komponentit voivat tarjota testipedin käyttöön metatietoa itsestään erityisen rajapinnan kautta
    - Tällöin testipetiä ei tarvitse etukäteen konfiguroida jokaista komponenttia/komponenttityyppiä varten, vaan se voi dynaamisesti testauksen aikana mukauttaa toimintaansa
    - Erikoisrajapinnat pidettävä tiukasti erillään normaaleista palvelurajapinnoista

Testattavuustaktiikat

- Testikohteen sisäisen tilan monitorointi
  - Testin kohde *ylläpitää suoritusaikana tietoa omasta tilastaan*, kuormituksestaan, käyttäjistään jne. ja tarjoaa pääsyn tähän dataan joko erityisen rajapinnan tai koettimen/jäljittimen kautta (monitor, probe, trace, log)
  - Monitorointi voi olla pysyvää tai tilapäistä (kytketään päälle ja pois käännös-/lataus-/suoritusaikana)
    - Hyödyllistä virheenjäljitystä varten
    - Testaus on yleensä tehtävä myös ilman monitorointia sen aiheuttaman yleisrasituksen vuoksi (overhead)
  - Myös erityiset Set/Get/Reset –operaatiot sisäisen tilan manipuloimiseksi ulkopuolelta ovat usein hyödyllisiä