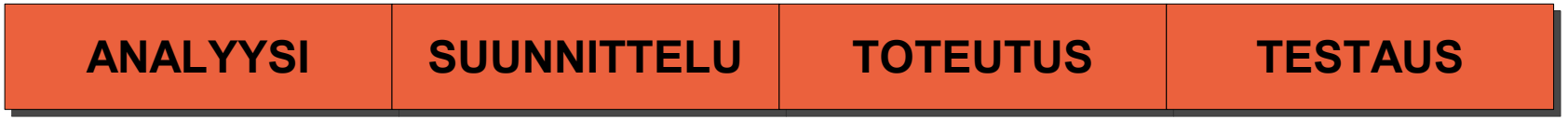
Ohjelmistojen ylläpito

JOT
7.12.2006
Minna Hillebrand

À la carte

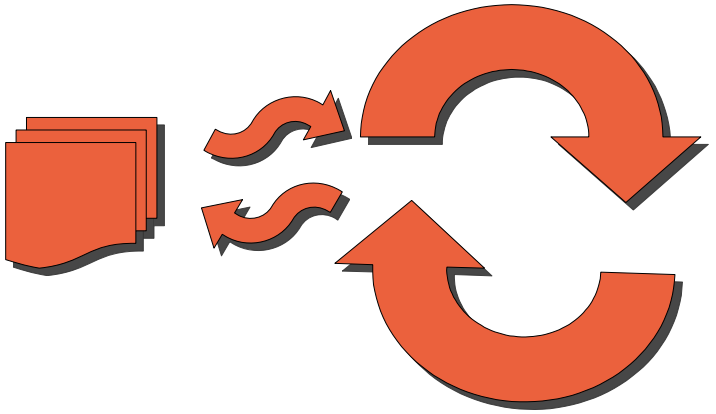
- Sovelluksen elinkaari
- Mitä tapahtuu “lopussa” ja miksi
- Ylläpidon muodot
- Uudistaminen
- Onko helpompaa tapaa?
- Korppi

Sovelluksen/projektin elinkaari



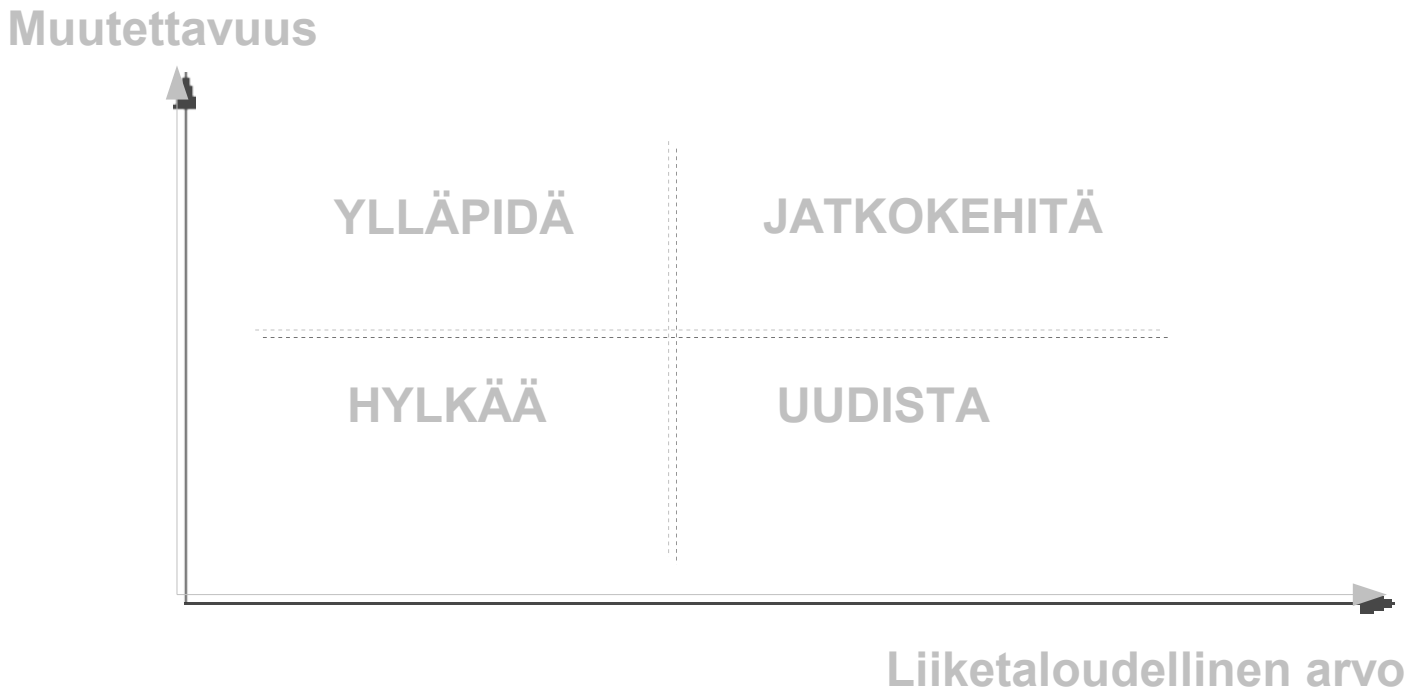
Sovelluksen oikea elinkaari

- Jatkuva iteraatio, jossa suunnittelu ja toteutus vuorottelevat. Ylläpito ei eroa oleellisesti muusta kehityksestä.



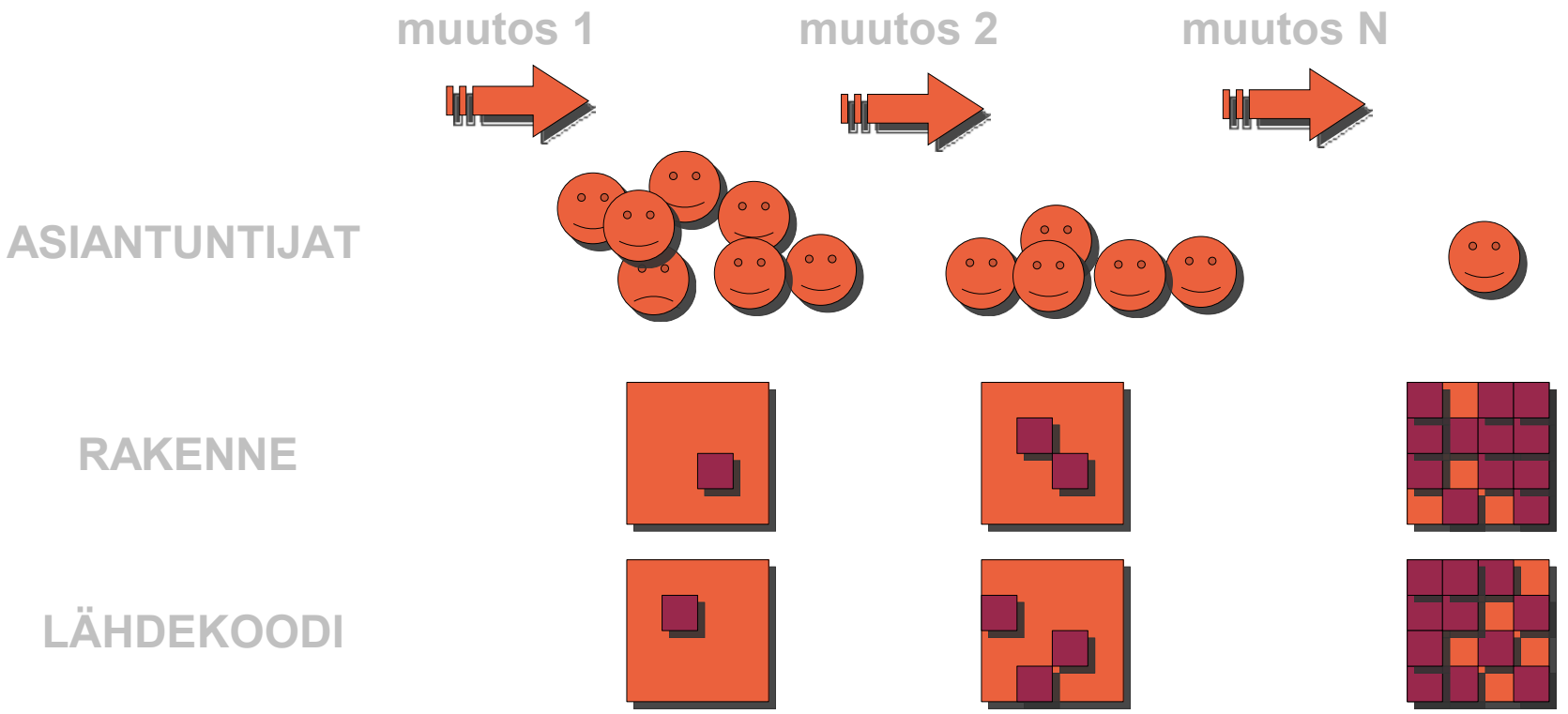
Kaikki hyvä loppuu aikanaan?

- Jatketaan, tai lopetetaan



Mitä tapahtuu sovelluksen ikääntyessä?

- Sovellus rapautuu



Miksi?

- Huono suunnittelu (arkkitehtuuri, toteutuksen yksityiskohdat, alusta...)
- Toistuvat muutokset
- Lehmanin lait

Lehmanin lait (1)

- Jatkuva muutos ja jatkuva kehitys jotta vastataan käyttäjien odotuksia; muutoin ohjelma jää pois käytöstä (1. ja 6., 1974 & 1978) Siten ei ole edes tarkoituksenmukaista suunnitella ohjelmaa, joka täyttää kaikki käyttötarpeet yhdellä kertaa.
- Monimutkaistuminen (2., 1974) vaatii vastavoimana aktiivista vastustamista. Muutoin sovelluksen laatu heikkenee (7., 1994).

Lehmanin lait (2)

- Muutosten määrä vakioituu ja säännönmukaistuu (5. ja 4., 1978) eli kehitysvauhti pysyy vakiona; aletaan pelkäämään kumuloituvia virheitä mikä hidastaa “normaalia” kehitysvauhtia.

Muuttujat

- Kehittäjät
- Ohjelmointityyli
- Ohjelmointikieli
- Tavoitteet

Ylläpidon muodot

- Korjaava ylläpito: korjataan löydetyt bugit ja siten parannetaan sovelluksen laatua, 20%
- Mukauttava ylläpito: laitteiston ja rajapintojen muutokset, 25%
- Täydellistävä ylläpito: tuottaa sovellukselle esitetyt uudet toiminnalliset vaatimukset, 50 %. Vaatii rakenteellisia muutoksia ja siten voi generoida uusia virheitä. Muuttaa sovelluksen toimintaa, joten myös testit on uusittava.

Ylläpidon muodot (2)

- Ehkäisevä ylläpito; ennakoidaan tulevaa ja parannetaan sisäistä rakennetta, <10%. Edellä mainitut eivät saa rapauttaa sovellusta nopeammin kuin tällä ehdotään korjaamaan.

Suunnan valinta

Sovelluksen uudistaminen

- Ohjelman rakenteen tulkitseminen.
- Sovellusta vastaavan dokumentaation tuottaminen ja ylläpito.
- Ohjelman rakenteen ja luettavuuden parantaminen; suunnittelu ja toteutus.
- Lopputuloksena ohjelmalla on parempi rakenne kuin lähtötilanteessa.

Uudistamisen vaiheet: 1

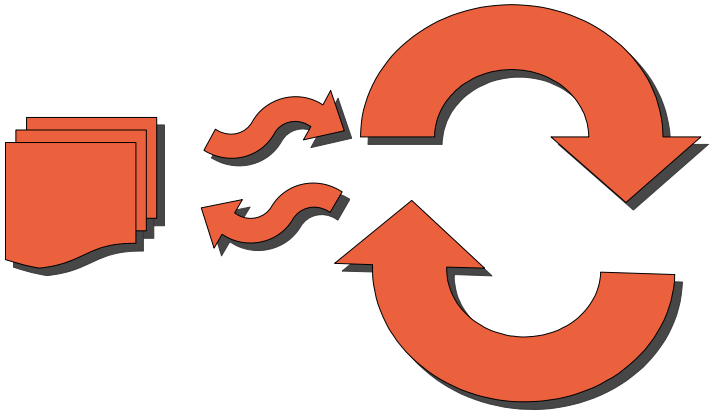
- Abstraktion luominen nykyisen rakenteen pohjalta takaisinmallintamalla (reverse engineering): UML-kaaviot
- Ei muuteta ohjelman tilaa, eikä lisätä uusia toimintoja! Rakennetaan testit, joilla voidaan varmentaa, että sovellus toimii odotetusti.

Uudistamisen vaiheet: 2

- Uudelleenmuokkaus eli uuden rakenteen suunnittelu riittävä suurella abstraktiotasolla (reengineering). Tunnettava haluttu lopputulos!!
- Yliluokkien lisääminen, toistuvien ohjelmanosien kapselointi, rajapintojen luominen ja/tai yksinkertaistaminen. Muutokset voidaan tehdä halutun lopputuloksen ehdoilla ja tulevat vaatimukset huomioiden
- Parantaminen (improvement), uusiminen (renewal), kohentaminen (refurbishing), ajanmukaistaminen sekä toimintojen että kehityksen tasolla (modernization)

Uudistamisen vaiheet: 3

- Suunnitelmien toteutus eli uudelleenrakentaminen (refactoring) kooditasolle.



Uudistamisen riskejä

- Työkaluja voi olla vaikea löytää.
- Käsityö tulee kalliiksi.
- Aika.
- Käsityössä virheiden todennäköisyys kasvaa .
- Asintuntijoita on vaikea löytää.
- Tavoitteet epäselviä.
- “ei trendikästä”
- Johdon hyväksynnän saavuttaminen.
- Positiivisia kokemuksia tarvitaan, jotta uskalletaan toistaa jatkossakin. Siksi ei ole varaa epäonnistua.
- Nämä riskit kannattaa ottaa.

Mitä on voitettavissa?

- Visualisoitu sovelluksen rakenne.
- Ajantasainen dokumentaatio.
- Kehitys jatkuu paremmalla alustalla.
- Työkaluja, joilla voidaan löytää vastaavat ongelmakohdat muissa kohteissa.
- Ammattitaitoa tekijöille ja osaamista yritykselle.

Muutosnopeus

- Yhtäkkinen uudistaminen (cold turkey). Koko sovellus uudistetaan erillään tuotantoversiosta ja käyttöönotetaan kerralla.
  - Kestää kauan ja maksaa paljon, ennen kuin tuloksia on nähtävillä
  - Muutokset sekä tuotantoon että uuteen versioon
  - Rajapintojen testaus? Datan siirto? Käyttöönotto? Muutosvastarinta?

Muutosnopeus (2)

- Vähittäisessä uudistamisessa (incremental approach) osia uudistetaan tarpeen mukaan.
  - Vaatii sovelluksen jakautumista sopiviin komponentteihin
  - Helpompi tuottaa ja testata
  - Käytössä on vanhan ja uuden hybridi
  - Voi vaatia lisäosia vanhan ja uuden järjestelmän välille
  - Voi venyä ajallisesti pidemmäksi
  - Ei voida muuttaa kokonaisrakennetta

Muutosnopeus (3)

- Kehittävä uudistaminen (evolutionary approach) etenee myös vähittäisesti, mutta sovelluksen toimintojen mukaan.
  - Sovelluksen toiminnalliset osat oltava tunnistettavissa
  - Saadaan modulaarinen sovellus
  - Sovelluksen sisäisesti yksittäinen komponentti voidaan uudistaa useammassa osassa
  - Rajapintojen muuttaminen hyvin vaikeaa
  - Kestää pisimpään

Periaatteita, takaisinmallinnus

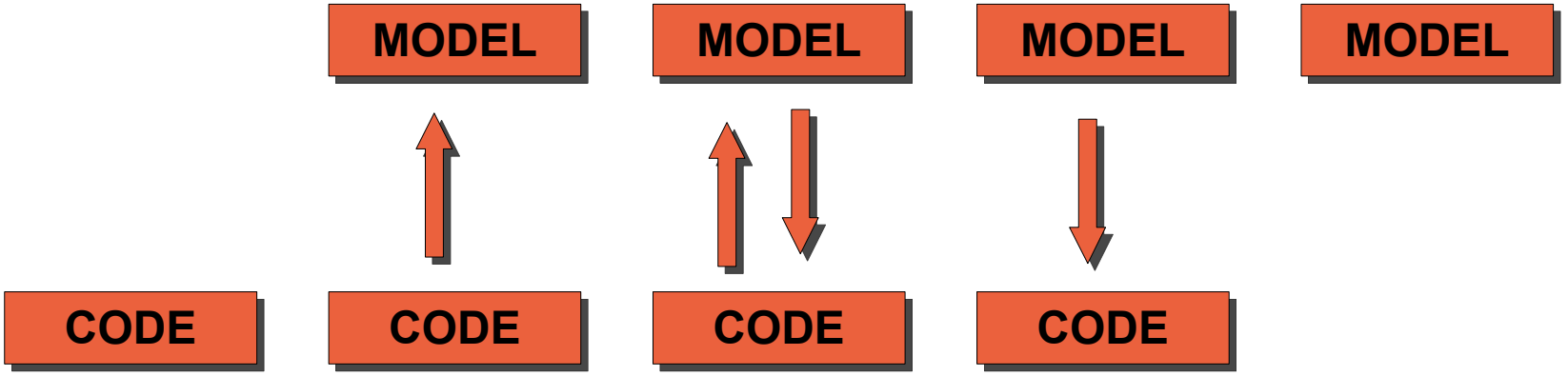
- Appoint a Navigation
- Agree on Maxims, Most Valuable First
- Fix Problems, Not Symptoms
- If It Ain't Broke, Don't Fix It
- Keep It Simple
- Read All The Code in One Hour, Skim the Documentation
- Study the Exceptional Entities
- Refactor to Understand

Periaatteita, muokkaaminen

- Write Tests to Understand
- Write Test to Enable Evolution
- Grow Your Test Base Incrementally
- Test the Interface, Not the Implementation
- Migrate Systems Incrementally
- Always Have a Running Version
- Deprecate Obsolete Interfaces
- Move Behavior Close to Data
- Split up God Class

Tulevaisuus?

- Model-driven development: lähdekoodi ja malli aina ajan tasalla.
- Abstraktiotaso nousee, kehittäminen on tuottavampaa.



Korpin ylläpito

- Asiakaspalvelua
- Virheiden korjaamista
- Uuden kehittämistä
- Lokien analysointia
- Suorituskyvyn parantamista
- Yhteensopivuus (css)
- Integrointi

Jakauma

- Neuvonta 30 %
- Tukipalvelut 10%
- Koodiin koskeminen 60 %

- Korjaava ylläpito 40 %
- Täydellistävä ylläpito 35 %
- Mukauttava ylläpito < 10 %
- Ehkäisevä ylläpito 20 %

Resurssit

- 6+3 kehittäjää
- Vastuulla 350 000 koodiriviä

Työvälineet, koodi

- Versionhallinta, CVS
- Kehitysympäristö, Eclipse
- Alusta, Linux / Windows
- Jaettu tietokanta, PostgreSQL

Työvälineet, prosessi

- Jaettu postiosoite korppi@jyu.fi
- Tracker
- Bugzilla (1470 todo / 1430 done)
- Wiki

- Prosessi?
- Välineet?

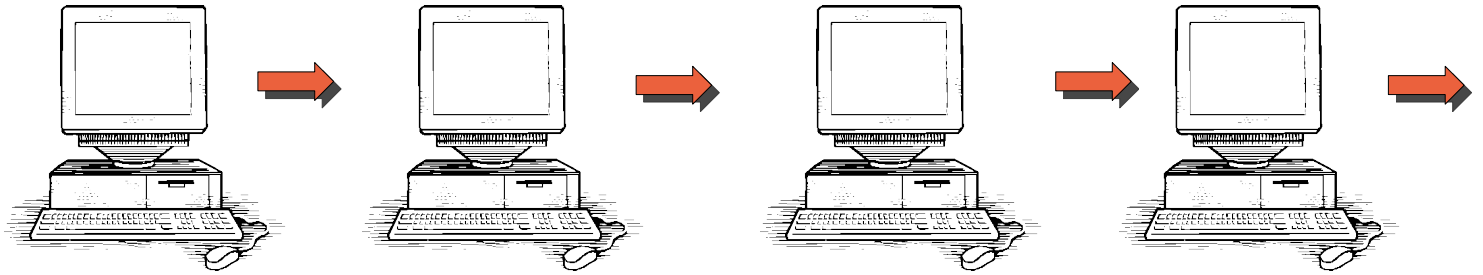
Sovelluksen käyttö

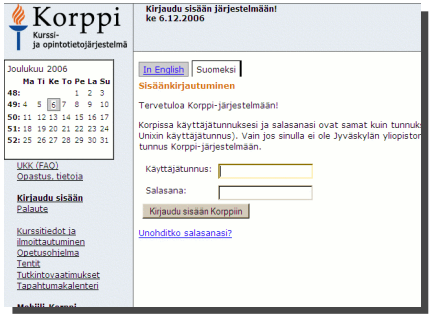
- 2 varsinaista palvelinta
- 2 mobiilipalvelinta
- 1 synkronointipalvelin
- 24 / 7 palvelun odotettu saatavuus, pois käytöstä vain päivitysten aikaan.
- Jatkuvasti vähintään 20 kirjautunutta käyttäjää
- Lokia generoituu giga / päivä, http://korppi.jyu.fi/statistics

- 06.12.2006 20:12:05 up 63 days, 5:55, 3 users, load average: 0.06, 0.14, 0.09
- ----------
- Kirjautuneena 53 käyttäjää
- 25.12.2005 04:12:01 up 14 days, 7:26, 1 user, load average: 0.02, 0.43, 0.38
- ----------
- Kirjautuneena 3 käyttäjää

Kehitysympäristö

- Lokaali kehitysympäristö
- Avoin ja yhteinen kehityspalvelin
- Avoin ja yhteinen testi- ja koulutuspalvelin
- Rajoitettu tuotantopalvelin
- Avoin tuotantopalvelin





Strategia

- Tehdään mitä pyydetään
- Tehdään paremmin (ehkä)

Tavallinen tarina

- Tietokantahaku
  - Aloitetaan kyselyllä, joka tekee tehtävänsä.
  - Joku valittaa hitaudesta.
  - Optimoidaan SQL-kyselyä.
  - Lisätään kantaan indeksejä.
  - Muokataan tulosjoukon käsittelyä.
- Ongelmia
  - Hitautta ei havaita pienellä datamäärällä.
  - Alikyselyissä IN vs. EXISTS riippuu mm. datamäärästä.
  - Tulosjoukon modifiointimahdollisuus teknisesti nuori.

Suorituskyky

- Tietokantahakuja tehdään niin vähän kuin mahdollista.
- Haun suoritus on mahdollisimman nopea.
- Systeemin sisällä suoraviivaisia, nopeita operaatioita.

Metriikat

- Kuinka paljon suoritusaikaa kuluu kullakin sivulla
- 1) tietokantaan
- 2) java-toimintaan
- Jos kumpi tahansa on suuri, toimintaa voidaan todennäköisesti optimoida
- 1 – 100 ms, 2 – 500 ms

Metriikat (2)

- Kuinka paljon suoritusaikaa on kulunut kunkin sivun suoritukseen? Voidaanko pisimpään kestäneitä suorituksia optimoida?
-
- Kuinka paljon kutakin sivua on pyydetty? Voidaanko useimmin pyydettyjä toimintoja optimoida?

Uuden kehitys

- Kehitystoiveita tulee useita per viikko. Osa pieniä, osa suurempia.
- Priorisointi käyttäjämassan ja poliittisten heijastusvaikutusten mukaan.
- Pieniäkin toiveita tulee toteutettua: ylläpitäjilläkin pitää olla jotain hauskaa tekemistä.

Mistä puhuttiin?

- Sovelluksen elinkaari
- Mitä tapahtuu sovelluksen ikääntyessä ja miksi
- Ylläpidon muodot
- Uudistamisen vaiheet, riskit, mahdolllisuudet
- Miten muutos tehdään
- Korpin ylläpito: resurssit, strategia

Kysymyksiä?