Verifiointi ja validointi

Verifiointi ja validointi (V & V)

- V&V:n tavoitteena on varmistaa, että
  - Ohjelmisto vastaa määrityksiä (verifiointi)
  - Ohjelmisto täyttää käyttäjän odotukset (validointi)
- V&V:ta tehdään koko ohjelmistoprosessin elinkaaren ajan
- Ohjelmiston
  - ei tarvitse olla virheetön (eikä se myöskään yleensä ole sitä)
  - täytyy olla tarkoitettuun käyttöön ”riittävän hyvä”
  - mitä on ”riittävän hyvä”, riippuu käyttöympäristöstä

V & V: vaikuttavia tekijöitä

- V & V: järjestelmän sopivuus tarkoitukseensa:

  1. Kuinka **kriittinen** ohjelmisto on?
  2. Millaiset **odotukset käyttäjillä** on ohjelmiston suhteen?
  3. **Kilpailutilanne**: kuinka nopeasti tuote pitää saada markkinoille?

Verifioinnin ja validoinnin ero

- **Verifioinnissa** varmistetaan, että ohjelmisto vastaa määrityksiään:

  ”Are we building the product right?”

- **Validoinnissa** varmistetaan, että ohjelmisto täyttää asiakkaan sille asettamat odotukset:

  “*Are we building the right product?”*

Tekniikat: tarkastukset

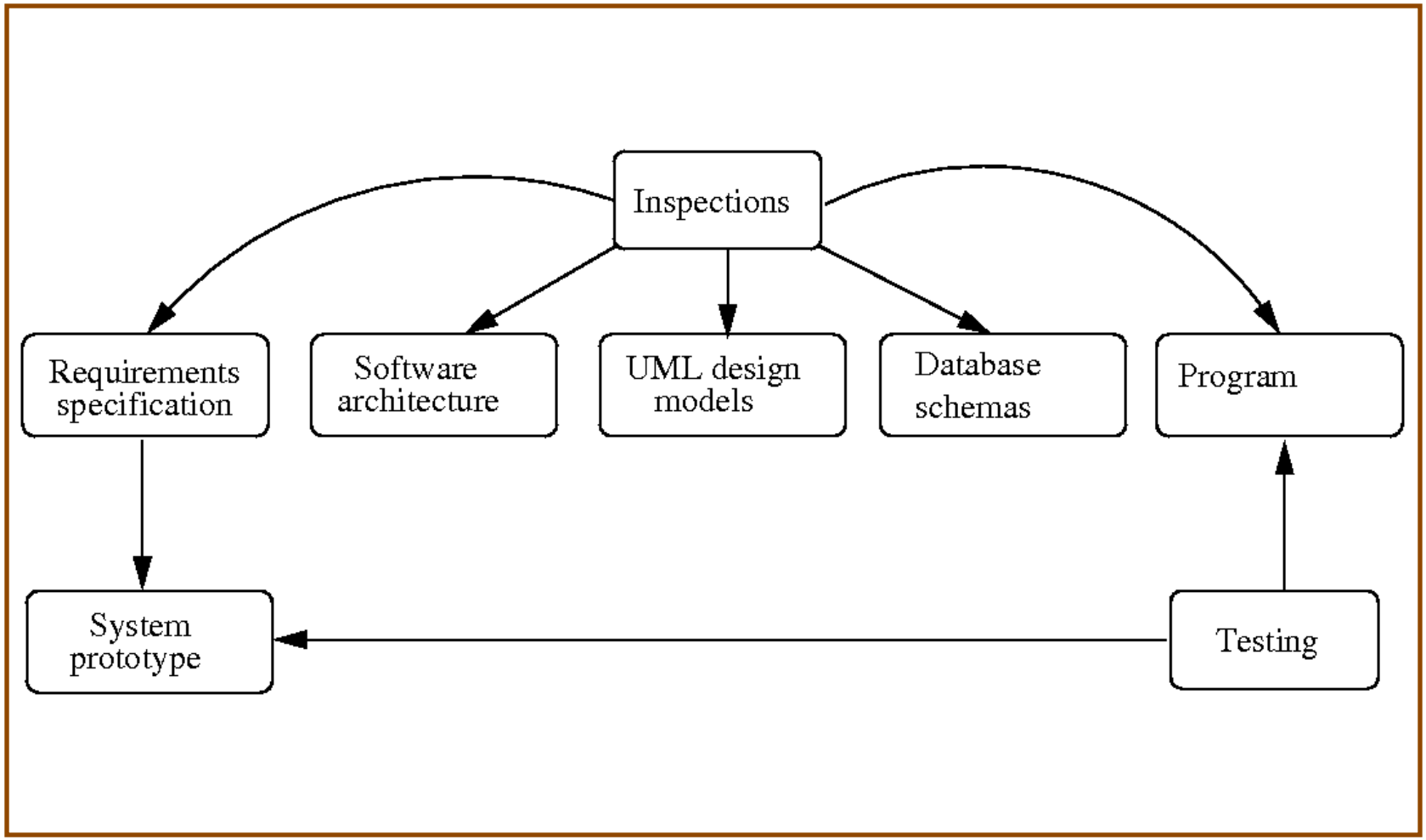
- V&V:ssa käytetään enimmäkseen kahta tekniikkaa: *tarkastuksia* ja *testausta*
- Tarkastuksissa (software inspections) isompi tai pienempi joukko ihmisiä analysoi ja tarkastaa järjestelmäkuvauksia muodollisessa tarkastustilaisuudessa
- Staattinen tekniikka: ei vaadi suorituskelpoista ohjelmaa
- Soveltuu käytettäväksi laaja-alaisesti esim. vaatimusdokumentin, suunnittelukaavioiden tai ohjelmakoodin verifiointiin ja validointiin

Tekniikat: testaus

- Testauksessa (software testing) ohjelmistoa tai sen osaa suoritetaan tietyillä testitiedoilla
- Tuloksia analysoimalla ja ohjelmiston suoritusta seuraamalla selvitetään, että toiminta on odotettua
- Dynaaminen tekniikka: tarvitaan suorituskelpoinen ohjelma

- Sekä tarkastuksia että testausta tarvitaan V&V:ssa: ne paljastavat eri tyyppisiä virheitä
- Mitä kriittisempi järjestelmä sitä enemmän tyypillisesti painopiste syytä olla tarkastuksissa

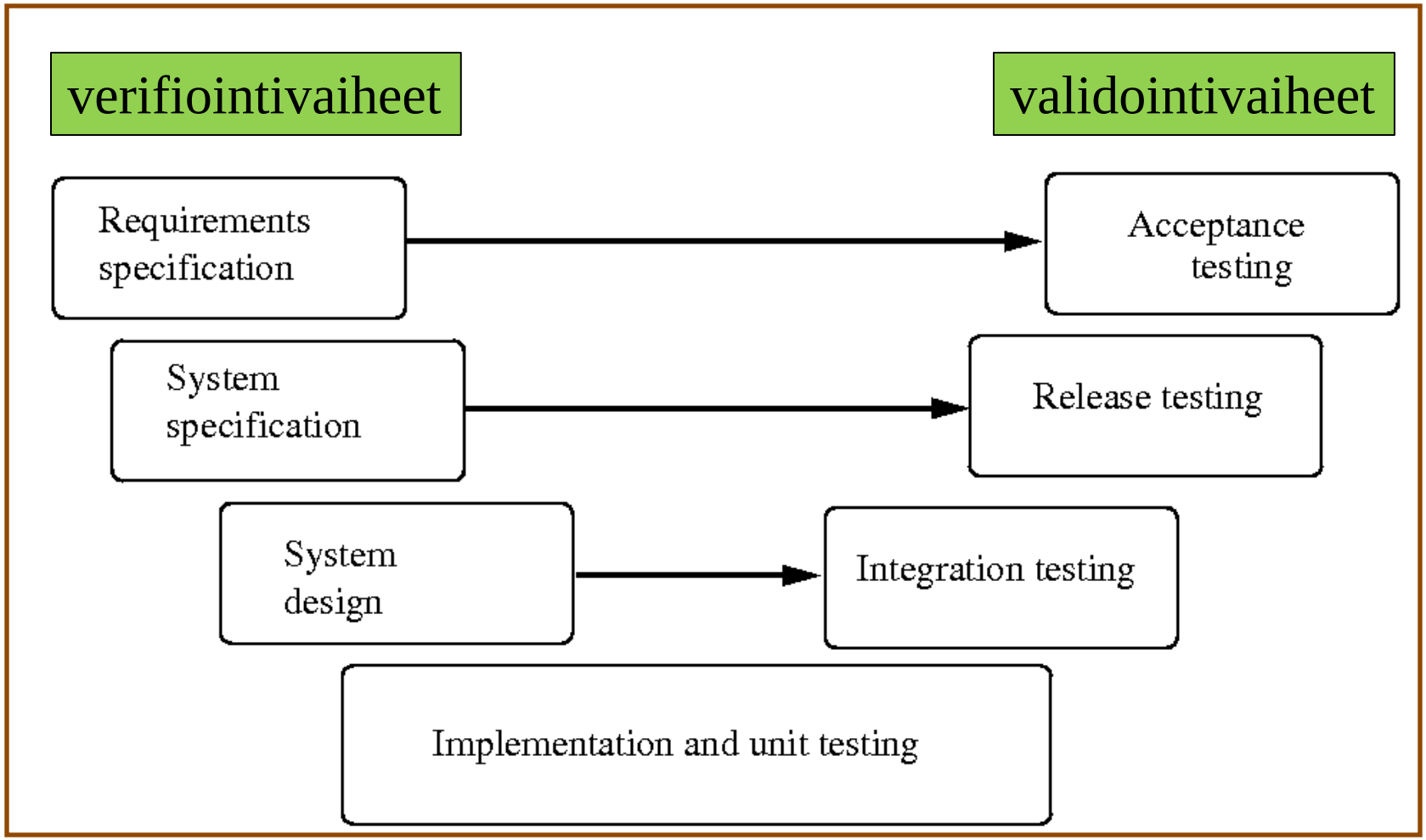
Tarkastukset ja testaus



V-prosessimalli

- V-malli on vesiputousmallin laajennos
- V-malli kuvaa V&V-työvaiheen suhteen muihin prosessin työvaiheisiin
- V&V-työvaihe mukana koko prosessin ajan
- Tarkastuksia voidaan pitää missä tahansa työvaiheessa tai niiden välillä

V-prosessimalli



Tarkastukset

Tarkastus

- Muodollinen kokous, jossa tarkastetaan jonkin projektin tuotos
  - puutteiden (vajaa määritys, puuttuva toiminta) ja virheiden (väärin tehty määritys, ei-toivottu toiminta) etsintä
- Tarkastus on *katselmointitekniikka* (review technique)
  - Katselmoinnissa yksi tai useampi henkilö etsii toisen tuotoksesta puutteita ja virheitä.
- Tehdään kaikissa prosessin työvaiheissa
- Mikä tahansa dokumentti voidaan validoida tarkastuksella
- Mitä aiemmin puute/virhe löydetään, sitä helpompi se on korjata.

Tarkastusten luonne

- Tarkastus on muodollinen tilaisuus, johon osallistuu 3-6 henkilöä
  - Tilaisuudella on tarkka aikataulu, enintään 2 h
  - Osallistujat edustavat eri sidosryhmiä asiakkaasta projektiryhmän jäseniin
- Tarkastuksessa keskitytään yksinomaan löytämään puutteita ja virheitä
- Tarkastukseen osallistujat eivät keskustele löydetyistä puutteista: sihteeri kirjaa ne ylös ja siirrytään eteenpäin
- Osallistujat valmistautuvat etukäteen tilaisuuteen tutustumalla tarkastettavaan dokumenttiin (noin 2 h)

Tarkastukseen osallistujat

- Osallistujien roolit:
  - Puheenjohtaja (moderator): vastaa tarkastuksen aikataulusta ja ohjelmasta
  - Sihteeri (scribe): kirjaa ylös esiin tulleet asiat
  - Alustaja (reader): kuvaa esitettävän asian
  - Kirjoittaja (author/owner): edustaa dokumentin tekijöitä
  - Tarkastaja (inspector): etsii dokumentista puutteita ja virheitä (kaikkien rooli)

Ennen tarkastusta

- Ennen tarkastustapahtumaa:
  - kaikki tarkastuksessa tarvittavat dokumentit ovat saatavilla,
  - osallistujilla on ollut aikaa tutustua dokumentteihin (valmistautuminen noin 2h),
  - osallistujat ovat saaneet tarkistuslistat yleisimmistä puutteista ja virheistä, ja
  - tarkastettava dokumentti on sellaisella tasolla, että siinä ei ole ilmeisiä virheitä

Tarkastuksen päätös

- Tarkastuksen lopuksi ryhmä äänestää tuotoksen hyväksymisestä:
  - Hyväksytään sellaisenaan: ei muutoksia.
  - Hyväksytään muutoksin: löydetyt puutteet ja virheet on korjattava, mutta tuotoksesta ei tarvita enää uutta tarkastusta.
  - Hylätään: löydetyt puutteet ja virheet on korjattava. Korjauksen jälkeen tuotoksesta käydään läpi uusi tarkastus.

Tarkastusten kultaiset säännöt

1. Arvioidaan tuotetta, ei tekijää
2. Suunnitellaan aikataulu ja pidetään siitä kiinni
3. Ei väittelyä
4. Ei ratkota löydettyjä ongelmia
5. Rajoitetaan osallistujien määrä 3-6 henkeen
6. Valmistaudutaan huolellisesti tarkastukseen
7. Käytetään tarkistuslistoja sekä valmistautuessa että tarkastuksessa
8. Varataan riittävästi aikaa ja resursseja
9. Koulutetaan osallistujat
   - Pidetään tarkastuksessa kännykät kiinni!

Testaus

Testaus

- Eri tarkoituksiin tarvitaan erilaisia testausstrategoita
- Testauksen tavoitteena voi olla
  1. Osoittaa sekä asiakkaille että projektille, että ohjelmisto täyttää sille asetetut vaatimukset
  2. Löytää ohjelmistosta puutteita ja virheitä, joiden takia ohjelmisto ei toimi, toimii väärin tai ei vastaa sille asetettuja määrittelyjä

- *Tapaus 1 → Validointitestaus*
  - Testitapaukset jotka vastaavat järjestelmän oletettua todellista käyttöä
  - Hyvä testitapaus näyttää järjestelmän toimivan toivotulla tavalla
- *Tapaus 2 → Vikatestaus*
  - Testitapaukset pyrkivät paljastamaan virheitä, eivät välttämättä heijastele järjestelmän oletettua normaalia käyttöä
  - Hyvä testitapaus paljastaa virheitä ja johtaa ohjelmiston laadun paranemiseen

Täydellinen testaus mahdotonta

- Täydellinen testaus, jossa ohjelma testataan kaikilla mahdollisilla syötteillä, syötekombinaatioilla ja ajoituksilla, ei ole käytännössä mahdollista
  - Jo hyvin yksinkertaisilla ohjelmilla kaikkien testitapausten suoritus veisi vuosia
- Tämän johdosta testauksessa valitaan osajoukko kaikista mahdollisista testitapauksista

Testauksen piirteitä

- Testauksen suunnittelu kannattaa tehdä V-mallin mukaisesti rinnan kehitystyön kanssa
- Valmistelu on testiympäristön suunnittelua, ohjelmointia ja tietojen keruuta
- Suoritus pitää automatisoida aina kun mahdollista, jotta testit voidaan suorittaa helposti uudelleen

Musta laatikko

- Musta laatikko (black box) –testaus
  - Ulkoinen näkökulma järjestelmään
  - Ei tietoa testattavan järjestelmän osan sisäiseen rakenteeseen
  - Testin suunnittelija valitsee sopivat syötteet ja määrittää oikean tuloksen
  - Käyttökelpoinen kaikissa testauksen vaiheissa

Valkoinen laatikko

- Valkoinen laatikko –testaus
  - White box testing, glass box testing
  - Testitapaukset suunnitellaan koodin sisäisen rakenteen tuntemuksen pohjalle
  - Eri suorituspolkujen tunnistaminen ja kunkin suorituspolun testaaminen
  - Virheen paikantaminen
  - Käytetään varsinkin yksikkötestauksessa

Oraakkeli

- ”Oikean” tuloksen määrittäminen välttämätöntä testauksessa
- Ei ole aina itsestään selvää, mikä olisi asetettava oikean suorituksen kriteeriksi
- Automatisoitu testaus: millä perusteella voidaan sanoa, että testi todella testaa sitä mitä sen pitäisi testata?
- *Oraakkeli* (oracle) on mekanismi, jonka perusteella määritellään onko suoritus oikea

Yksikkö- ja komponenttitestaus

- *Yksikkö-* ja *komponenttitestaus* tehdään ennen *järjestelmätestausta* (johon kuuluvat *integrointitestaus* ja *julkistustestaus*)
- Yksikkö ja komponenttitestaus kuuluvat pääosin ohjelmiston kehittäjille (projektiryhmälle)
- **Yksikkötestaus** (unit testing)
  - Yksittäisten olioiden ja metodien toiminnan oikeellisuuden varmistaminen
  - Yksikkötestaus ja koodaus ovat sidoksissa: koodatessa tehdään myös yksikkötestejä

- **Komponenttitestaus**
  - Komponentin toteuttavat oliot integroidaan ja testataan jakamattomana kokonaisuutena
  - Olioiden ja komponenttien palvelut määritellään rajapintojen kautta
  - *Rajapintatestaus*: etsitään virheitä, jotka johtuvat rajapintojen virheistä tai vääristä rajapintojen oletuksista

Interface testing guidelines (Sommerville 2010)

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
- Always test pointer parameters with null pointers.
- Design tests which cause the component to fail.
- Use stress testing in message passing systems.
- In shared memory systems, vary the order in which components are activated.

Integrointitestaus

- Integrointitestauksessa testataan valmiiden yksittäin toimivien komponenttien **yhteistyö** osajärjestelmässä
- Virheiden paikantamisen kannalta on tärkeää tehdä integrointi inkrementaalisesti
  - kuhunkin osajärjestelmään liitetään komponentteja yksi kerrallaan ja testataan komponenttien yhteistyö
- Black box ja white box –strategiat käyttökelpoisia
- Integrointia voidaan tehdä
  - ylhäältä alas (top-down), jolloin aloitetaan kokoaminen järjestelmän kokonaisuutta ohjaavista osista
  - alhaalta ylös (bottom-up), jolloin aletaan kokoamaan suorittavia osia

Julkistustestaus

- Kehittäjätiimin ulkopuolella käytettäväksi tarkoitetun järjestelmäversion testaus (Sommervillella erikseen ‘julkistustestaus’, *release testing*, ja käyttäjien tekemä ‘hyväksymistestaus’ *acceptance testing*)
- Testauksen tulee osoittaa, että järjestelmä on riittävän hyvä käytettäväksi:
  - täyttää sille asetetut vaatimukset toiminnallisuuden, suorituskyvyn ja luotettavuuden suhteen
- Yleensä black box –testaus
- Testit johdetaan järjestelmäkuvauksista
- Erillinen testaustiimi, testaajilla ei tietoa toteutuksesta

Julkistus- ja järjestelmätestaus

- Julkistustestaus on järjestelmätestauksen osa
- Kehittäjätiimistä erillisen tiimin pitäisi toteuttaa julkistustestaus

  → black box -asetelma
- Kehittäjätiimin tekemän järjestelmätestauksen painopiste virheiden ja puutteiden etsimisessä (vikatestauksessa)
- Julkistustestauksen päämääränä varmistaa järjestelmän soveltuvuus käyttäjätiimin ulkopuoliseen käyttöön (painopiste validointitestauksessa

Regressiotestaus

- Regressiotestauksella varmistetaan, etteivät järjestelmään tehdyt muutokset ole “rikkoneet” aiemmin toimivaa koodia
- Manuaalisessa testausprosessia regressiotestaus on kallista
- Automatisoidussa testauksessa suoraviivaista: jokaisen muutoksen jälkeen kaikki testit ajetaan uudestaan
- Muutos hyväksytään vasta kaikkien testien läpimenon jälkeen

Suorituskykytestaus

- Integrointitestauksen jälkeen osana julkistustestausta testataan järjestelmän kriittisiä ei-toiminnallisia ominaisuuksia
  - suorituskykyä
  - luotettavuutta ja vikasietoisuutta
  - turvallisuutta
- Usein sarja testejä, joissa järjestelmän kuormaa asteittain kasvatetaan
- *Rasitustestauksessa* (stress testing) ohjelma viedään äärirajoille ja vielä niiden ylikin

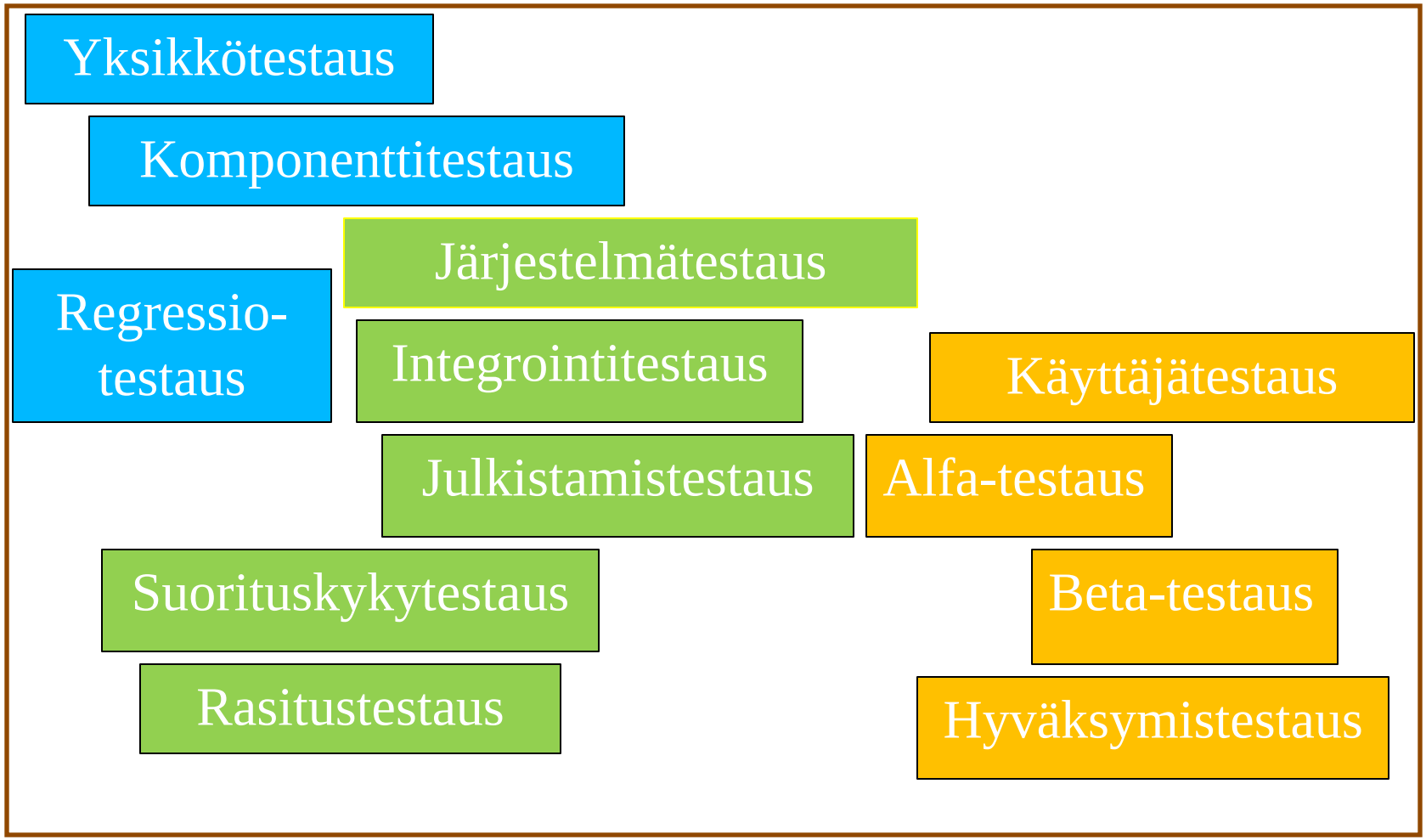
Käyttäjätestaus

- Käyttäjät testaavat järjestelmää
- Oleellinen osa testausta, vaikka huolellinen järjestelmä- ja julkistustestaus olisikin jo toteutettu
  - Käyttäjien työskentely-ympäristöön liittyvät tekijät, joita ei voida toistaa testausympäristössä
  - Käytettävyys, luotettavuus, suorituskyky

Käyttäjätestauksen tyyppejä

- Alfa-testaus
  - Käyttäjät testaavat yhdessä kehittäjätiimin kanssa
- Beta-testaus
  - Ohjelmistosta julkistetaan versio käyttäjille itsenäisesti kokeiltavaksi
- ***Hyväksymistestaus***
  - Käyttäjät testaavat järjestelmää käyttöympäristössä, arvioidaan voidaanko järjestelmä hyväksyä käyttöön

Testauskäsitteiden yhteyksiä



Testivetoinen ohjelmistokehitys

- Testaus ja toteutus tiukasti liittyneinä toisiinsa
- Mahdollisimman pitkälle automatisoitu testaus

- Yksikkötestien kirjoittaminen ennen testattavien yksikköjen toteutusta
- Inkrementaalisuus: testipatteristo kasvaa toteutuksen myötä (regressiotestaus)
- Refaktoroinnin (XP) yhteydessä yksikkötestit aina läpäistävä
- Kaksiarvoisuus: testi joko läpäistään tai ei (pass or fail) → ei manuaalista testitulosten analysointia

Esimerkki: testin kirjoittaminen ennen toteutusta

```
public class MoneyTest extends TestCase {

public void testSimpleAdd() {
   Money m1 = new Money(12, ”euro”);
   Money m2 = new Money(14, ”euro”);
   Money expected = new Money(26, ”euro”);
   Money result = m1.add(m2);
   assertEquals(expected,result);
   }
}
```

Lähde: Craig Larman: Agile & Iterative Development. A Manager’s Guide.

Testivetoinen ohjelmistokehitys

- Hyväksymistestit suunnitellaan asiakkaan kanssa, ei erillistä hyväksymistestausprosessia
- Hyväksymistestit liittyvät käyttäjäkertomuksiin
- Hyväksymistestit määrittävät sen mitä järjestelmän hyväksymiselle tarkoitetaan

→Vaatimusten ja testauksen läheinen yhteys

- Testivetoisen ohjelmistokehityksen periaatteita voidaan yhdistää myös suunnittelukeskeiseen ohjelmistokehitykseen

Testivetoinen kehitys: testaus+toteutus

1. Lisätoiminnallisuuden määrittäminen
   - Pitää olla pieni ja toteutettavissa muutamalla koodirivillä
2. Automatisoidun testin määrittäminen ja kirjoittaminen uudelle toiminnallisuudelle
3. Testin ajaminen yhdessä kaikkien muiden testien kanssa
   - Ennen kuin uutta toiminnallisuutta on koodattu, testi ei saa mennä läpi
4. Uuden toiminnallisuuden toteuttaminen ja testin ajaminen
5. Kun kaikki testit läpäisty, siirrytään toteuttamaan seuraavaa toiminnallisuutta

Etuja

- Testaus ei jää tekemättä!
- Testaus tukee suunnittelua: testattavan yksikön toiminta käydään läpi huolellisesti ennen toteutusta
- Toteutus ei vaikuta testaukseen (sekä etu että haitta)
- Kaikelle koodille on vähintään yksi testi
- Testit toimivat dokumentaationa siitä, mitä järjestelmän (osien) pitäisi tehdä
- Virheenjäljittämisen yksinkertaistuminen
- ”Semi-enjoyable way to do testing”

-Craig Larman, Agile & Iterative Development

Haittoja

- Millaisia huonoja puolia testivetoisessa ohjelmistokehityksessä on?

Ohjelmiston evoluutio

- Ohjelmiston käyttöönoton jälkeen se on vasta elinkaarensa alussa
- Syntyy **muutospaineita**, koska
  - ohjelmistolle asetetaan uusia vaatimuksia
  - vanhat vaatimukset muuttuvat
  - ohjelmistosta löytyy virheitä
  - ohjelmisto täytyy saada toimimaan uudessa ympäristössä

  → Ohjelmiston evoluutio

Ohjelmiston ylläpito

- Prosessia, jossa ohjelmiston muutospaineita helpotetaan ohjelmistoa muokkaamalla , kutsutaan *ohjelmiston ylläpidoksi* (software maintenance)
- Noin 50-90% ohjelmistoon kuluvista resursseista tarvitaan ohjelmiston luovuttamisen jälkeen

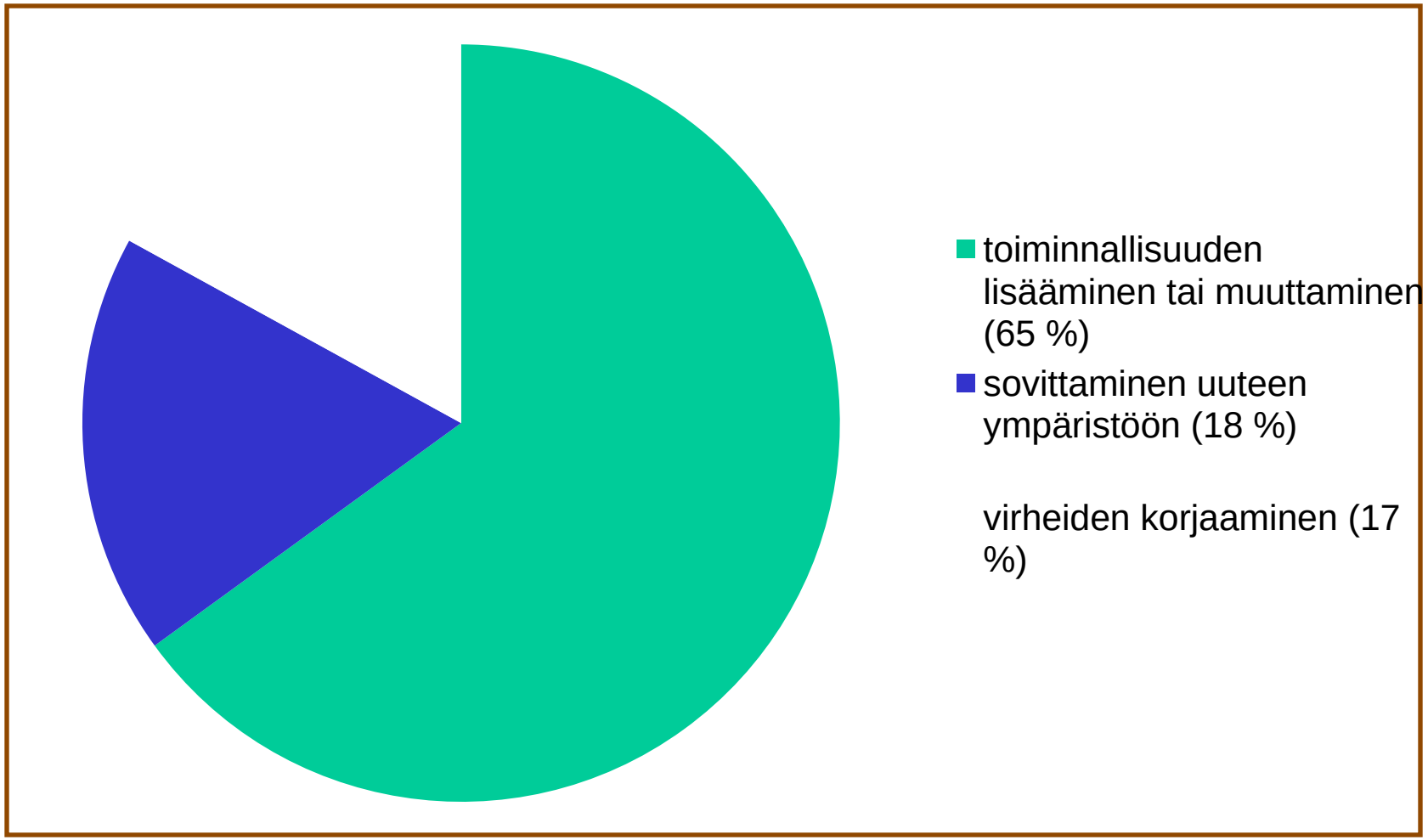
Ohjelmiston ylläpito

- Ylläpito on yleinen termi prosessille, jonka avulla jo luovutettua ohjelmistoa muutetaan sen elinkaaren aikana
- Ylläpidosta on yleensä vastuussa erillinen ylläpitoryhmä
- Ylläpidossa tehdään yleensä kohtuullisen pieniä muutoksia
- Ohjelmiston arkkitehtuuri ei muutu. (Evoluutiossa se voi muuttua)

Ylläpitotyypit

- Ylläpitoa on kolmea eri tyyppiä:
  1. Ohjelmistovirheiden korjaus (korjaava ylläpito)
  2. Ohjelmiston sovittaminen uuteen ympäristöön (sovittava ylläpito)
  3. Ohjelmiston toimintojen muokkaus tai uusien toimintojen lisäys (täydentävä ylläpito)
- Usein ohjelmiston muutos vaatii kaikkia kolmea ylläpitotyyppiä

Ylläpitotyyppien osuudet



Evoluutioprosessi

- Ohjelmiston evoluutioprosessi vaihtelee huomattavasti. Prosessiin vaikuttaa
  - ylläpidettävän tuotteen tyyppi,
  - kehitystyössä käytetty prosessi ja
  - ylläpitoon osallistuvien henkilöiden ammattitaito
- Evoluutioprosessi voi vaihdella täysin epämuodollisesta prosessista hyvin tarkasti ohjattuun prosessiin

Evoluutioprosessin vaiheet

- Muutostarpeiden tunnistaminen
- Vaikutusanalyysi (impact analysis)
  - Selvitetään, miten paljon muutos vaikuttaa ohjelmiston rakenteeseen ja mitä sen toteutus tulisi maksamaan
- Julkaisusuunnittelu (release planning)
  - Päätetään, mitä muutoksia seuraavaan ohjelmistoversioon tehdään. Muutokset aiheuttavat jonkun kolmesta ylläpitotyypistä
- Päivityksen toteutus
- Version julkaisu (system release)

Evoluutio- ja kehitystyöprosessit

- Evoluutioprosessi ei ala tyhjästä
  - Ensimmäisenä vaiheena on ymmärtää jatkokehitettävän ohjelmiston rakenne ja tehdyt ratkaisut
  - Muutokset tulee tehdä sellaisiksi, että ne eivät riko olemassaolevia ratkaisuja.
- Vaihtoehtona on *ohjelmiston uudistaminen* (software re-engineering)

Ketterät menetelmät ja evoluutio

- Inkrementaalisuus

→Raja kehittämisen ja ylläpidon välillä ei yhtä selvä kuin perinteisessä ohjelmistokehityksessä

- Ylläpito jatkoa tiheille versioiden julkistamiselle
- Automatisoitu regressiotestaus: erityisen arvokasta kun järjestelmään tehdään muutoksia
- Muutosten ilmaiseminen uusina (tai päivitettyinä) käyttäjäkertomuksina

- Ongelmia voi tulla, jos kehitystyö on tehty ketterillä menetelmillä, mutta ylläpidossa suunnittelukeskeinen lähestymistapa
  - Yksityiskohtaisen dokumentaation puute
- Ongelmia voi tulla jos kehitystyö on tehty suunnittelukeskeisesti, mutta ylläpito ketterillä menetelmillä
  - Automatisoidun testauksen kehittäminen

Ohjelmiston uudistaminen

- Jos näyttää siltä, että ohjelmiston ylläpidon kustannukset alkavat kohota taivaisiin eikä silti voida taata laadukasta lopputulosta, voi olla aika uudistaa ohjelmisto.
- Ohjelmiston uudistamisessa osa ohjelmistosta toteutetaan ja dokumentoidaan uudestaan sellaiseksi, että sen ylläpito helpottuu

Ohjelmistojen uudistaminen

- Ohjelmiston uudistamisessa ohjelmiston toiminnallisuus ei muutu
- Ainoastaan toiminnallisuuden toteuttavat ratkaisut muuttuvat
- Loppukäyttäjälle ohjelmisto näyttää ja (yleensä) tuntuu samalta
- Ohjelmisto saattaa myös tehostua uudistamisessa, jolloin ohjelmiston rajoitukset ja reunaehdot lievenevät. Sen sijaan rajoitukset ja reunaehdot eivät saa tiukentua

Uudistamisen ja kehitystyön ero

- Ohjelmiston uudistaminen eroaa kehitystyöstä siinä, että uudistettavasta ohjelmistosta tiedetään jo vaatimukset. Näin vaatimusmäärittely jää pois.
- Lisäksi vanhan järjestelmän suunnitelmasta voidaan käyttää osia, jolloin suunnittelu helpottuu.
  - Esimerkiksi ohjelmiston yleisarkkitehtuuri ei yleensä muutu uudistuksessa.

Perinnejärjestelmät

- *Perinnejärjestelmä* (legacy system) on vanhentunut ohjelmisto, jota käytetään edelleen, mutta jonka suunnittelusta ja toteutuksesta ei enää tiedetä juuri mitään.
- Perinnejärjestelmä on tavallaan mennyt yli elinkaarestaan. Järjestelmää käytetään vaihtelevista syistä kuitenkin edelleen.

Perinnejärjestelmän evoluutiostrategiat

- Jossain vaiheessa perinnejärjestelmää käyttävä yritys joutuu päättämään, mitä järjestelmälle tehdään:
  - Järjestelmä voidaan hylätä
  - Järjestelmälle voidaan antaa tekohengitystä ylläpidon keinoin
  - Järjestelmä voidaan uudistaa
  - Järjestelmä voidaan korvata uudella vastaavalla järjestelmällä

Lehmanin lait

- Ohjelmiston evoluutio on muutoksen hallintaa. Työvaiheesta Lehman ja Belady kehittivät joukon lakeja, joita kutsutaan *Lehmanin laeiksi*
- Selittävät ohjelmiston muutospaineita ja niiden vaikutuksia ohjelmistoon ja sidosryhmiin
- Lakeja ei ole todistettu, joten ne ovat oikeastaan teoreemoja (otaksumia)

Lehmanin 1. laki

- Jatkuvan muutoksen laki:
  - Todellisessa käytössä olevan ohjelmiston täytyy joko muuttua tai ajan myötä menettää hyödyllisyyttään käyttöympäristössä
- Lain mukaan ohjelmiston evoluutiota ei voida välttää. Ohjelmistot tulevat aina tarvitsemaan muutoksia elinkaarensa aikana

Lehmanin 2. laki

- Kasvavan monimutkaisuuden laki:
  - Ohjelmistoa muutettaessa sen rakenne muuttuu entistä monimutkaisemmaksi (ja siten virhealttiimmaksi). Kehitys voidaan pysäyttää, mutta se vaatii resursseja.
- Laki on eräänlainen ohjelmiston entropian laki. Sen mukaan ajan myötä järjestelmän rakenne hajoaa, jos kehitykseen ei varauduta lisäresurssein

Lehmanin 3. laki

- Ison ohjelmiston evoluution laki:
  - Ohjelmiston evoluutio ohjaa itse itseään. Ohjelmiston koko, julkaisuaika, raportoitujen virheiden määrä pysyvät samassa suuruus-luokassa koko ohjelmiston elinkaaren ajan.
- Tämä laki on *ylläpidon ajankäytön laki*. Sen mukaan isojen ohjelmien ylläpito määräytyy jo kehitystyövaiheessa.

Lehmanin 4. laki

- Kehitystyöryhmän stabiilius:
  - Kehitystyön tehokkuus pysyy likimain vakiona ohjelmiston evoluutiovaiheen ajan.
- Lehmanin 4. lain mukaan ohjelmiston evoluutiota ei voida nopeuttaa.
- Yhdessä 3. ja 4. laki sanovat, että evoluutio on jokseenkin riippumaton hallinnan päätöksistä. Evoluutio on eräänlainen ohjelmistojen luonnonlaki.

Lehmanin 5. laki

- Säilytyksen laki:
  - Ohjelmiston elinkaaren aikana jokaisen uuden version esittelemät muutokset ovat likimain samansuuruiset verrattuna esittelytiheyteen.
- Lain mukaan isot muutokset ohjelmistoon synnyttävät paljon virheitä.
- Virheiden korjaamiseen menee aikaa; seuraava uusi versio tulee myöhemmin.

Lehmanin loput lait

- Viimeiset Lehmanin lait käsittelevät sitä, miten asiakkaat näkevät ohjelmiston evoluution. Ne voidaan tiivistää seuraavaan sanomaan:
  - mitä vähemmän ohjelmistoa ylläpidetään, sitä tyytymättömämmiksi ohjelmiston käyttäjät tulevat ajan kanssa.
- Lehmanin lait kuvaavat varsin hyvin *isojen räätälöityjen ohjelmistojen evoluutiota.*