6. Suunnitelmakeskeiset prosessit (lukuisia lähteitä)

- Ennen ketteriä prosessimalleja kehitettyjä prosesseja kutsutaan nykyisin *suunnitelmakeskeisiksi* (plan-driven) prosesseiksi.
  - Suunnitelmakeskeisyys tarkoittaa, että prosesseissa tehdään paljon valmistelevaa työtä ennen varsinaisen toteutuksen alkua.
  - Suunnitelmakeskeisyys ei tarkoita lineaarisuutta. Esimerkiksi spiraalimalli on sekä iteratiivinen että suunnitelmakeskeinen.
  - Suunnitelmakeskeiseen prosessiin perustuvat projektit voivat vallan hyvin onnistua. Tärkeintä on löytää projektille oikea prosessi. Tämä taas riippuu ainakin siitä, miten isosta projektista on kyse, miten muuttuvia projektin vaatimukset ovat, miten paljon tarvitaan dokumentaatiota ja miten muodollista laadunvarmistusta tarvitaan.

Vesiputousmalli

- *Vesiputousmallia* (waterfall model) pidetään ensimmäisenä varsinaisena prosessimallina. Sen isänä pidetään Winston Roycea, joka 1970 kirjoitti artikkelin ”Managing the Development of Large Software Systems” (Royce, 1970).
  - Royce kyllä esitteli vesiputousmallin periaatteet paperissaan, mutta Roycen mukaan ohjelmistotuotantoa *ei missään nimessä* pidä tehdä mallin mukaan.
  - Samassa artikkelissa Royce suosittelee mallia, missä kolmasosa ajasta käytetään prototyypin tekoon ja kaksi kolmasosaa proton perusteella tehtävän tuotteen tekoon.
  - Kuitenkin hyvin pian Roycen artikkeliin viitattiin vesiputousmallin isänä, vaikka Royce todellisuudessa kannatti iteratiivista kehitystyötä (Larman, 2003).

Vesiputousmallin suosio

- Vesiputousmalli saavutti nopeasti suurta suosiota, sillä se vastasi oman aikansa ohjelmistoammattilaisten käsitystä oikeasta prosessista.
- Malli on käytännössä vastaava kuin järjestelmäsuunnittelun malli muissa insinööritieteissä. Sen avulla on helppo suunnitella siltoja, mutta useimpien ohjelmistojen tekoon se on liian kankea.

Vesiputousmallin suosio - 2

- Mallin kömpelyydestä huolimatta suuri osa nykyisistäkin ohjelmistoyrityksistä käyttää vesiputousmallia tai sen varianttia kehitystyössään.
- Mallin eduksi on sanottava, että valtava osa viimeisen 40 vuoden ohjelmistotekniikan tutkimustyöstä perustuu vesiputousmalliin. Mallin teoria, vahvuudet ja heikkoudet tunnetaan hyvin.
- Kaikista malleista vesiputousmalli tarjoaa parhaan teoria- ja työkalutuen.

Vesiputousmallin perusteet

- Vesiputousmalli jakaa prosessin lineaarisiin vaiheisiin. Edellisen vaiheen tulos on seuraavan vaiheen syöte.
- Alkuvaiheessa lineaarisuus oli ehdotonta, mutta nykyisin myös vesiputousmallin mukaisissa prosesseissa sallitaan pientä iteratiivisuutta.

Vesiputousmallin vaihejako

- Modernin vesiputousmallin mukaisen prosessin vaiheet ovat seuraavat (Kan, 2003 mukaillen):
  - *Vaatimusmäärittely* (requirements gathering and analysis)
  - *Arkkitehtuurisuunnittelu* (architectural design)
  - *Suunnittelu (design),* jota voidaan tehdä rinnakkain arkkitehtuurin osille ja joka sisältää seuraavat vaiheet:
    - *Korkean tason suunnittelu* (high-level design / specification)
    - *Matalan tason suunnittelu* (low-level design)
    - *Suunnitelmien tarkastukset* (inspections)
    - *Koodaus* (coding)
    - *Kooditarkastukset* (code inspections)
    - *Yksikkötestaus* (unit testing)
  - Testaus (testing), joka sisältää seuraavat vaiheet:
    - *Integrointitestaus* (integration)
    - *Komponenttitestaus* (component testing)
    - *Järjestelmätestaus* (system testing)
    - *Hyväksymistestaus* (acceptance testing)

Vesiputousmalli ja dokumentaatio

- Vesiputousmalli perustuu hyvään dokumentaatioon ja tuotosten (dokumenttien) huolelliseen tarkastamiseen.
- Kun tuotos on tarkastettu, se *jäädytetään* (freeze). Jäädytettyä dokumenttia ei saa muuttaa ilman projektin johtoryhmän lupaa.
- Dokumenttien jäädytys tarkoittaa käytännössä, että vesiputousmallin projekteissa vaatimukset on kerättävä kerralla.
  - Uudet ja muuttuvat vaatimukset muuttaisivat jäädytettyjä dokumentteja.

Vesiputousmallin edut

- Malli on selkeä ja helposti omaksuttava.
- Mallille on kehitetty laaja teoria- ja työkalutuki.
- Malli ei vaadi projektin osallistujilta erityistaitoja.
- Malli sopii hierarkkisiin organisaatioihin.
- Malli ei rajoita tai vaadi käytettäviä tekniikoita, joten se on aika vapaasti räätälöitävissä.
- Malli toimii myös sellaisten tekijöiden kanssa, jotka eivät halua jatkuvaa ryhmätyötä ja isoja projektipalavereja.
- Mallille on kehitetty toimiva laadunvarmistus.
- Mallille on kehitetty toimiva prosessin parannus.
- Asiakas saa selkeän ja ymmärrettävän kustannusarvion jo projektin alussa.
  - Mutta! Kustannusarvio on usein väärä. Projektin alussa tehty arvio voi heittää nelinkertaisesti verrattuna todellisuuteen.
- Kiireisen asiakkaan ei tarvitse osallistua projektityöhön.

Vesiputousmallin haitat

- Vaatimukset eivät saa muuttua!
  - Jos tämä vaatimus ei toteudu, vesiputousmallia ei kannata käyttää.
- Asiakas näkee valmista vasta projektin päättyessä.
- Aikataulun lipsuessa testaus kärsii.
- Projektityöntekijät saattavat turhautua jatkuvaan dokumentointiin ja tarkastuksiin.
- Malli ei suosi luovuutta.
- Väärille urille joutuneen projektin saattaminen oikeaan suuntaan on vaikeaa.
- Myöhässä olevaa projektia on vaikeaa saada takaisin aikatauluun.
  - Iso osa vesiputousmallin mukaisista projekteista epäonnistuu tästä syystä: projektit eivät valmistu aikataulussa.

V-malli

- *V-malli* (V-model) on puhdasta vesiputousmallia kehittyneempi lineaarinen malli.
- V-mallia ajatellaan yleensä verifioinnin ja validoinnin mallina, mutta malli määrittelee kyllä puhtaan lineaarisen vaihejaon muiden prosessimallien tapaan.

V-malli 2

- V-malli perustuu Myersin ideoihin (Myers, 1979). Ensimmäinen varsinainen V-malli esiteltiin Saksassa 1986 ja Yhdysvalloissa hiukan myöhemmin Malli yleistyi 1990-luvun alkupuolella.
- V-malli elää ja voi hyvin. Siitä on esitelty useita variantteja, ja se on suosittu erityisesti Keski-Euroopassa
  - (http://v-modell.iabg.de/).

V-Mallin perusteet

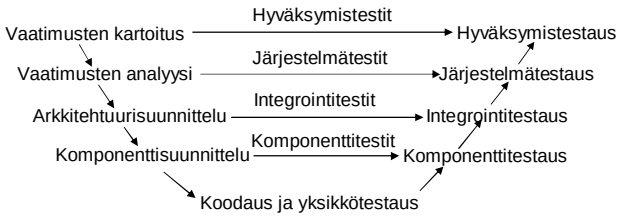
- *V-malli* jakaa kehitystyön kahtia:
  - suunnitteluun ja toteutukseen,
  - testaukseen.
- Mallin idea on siinä, että jokaista suunnittelu- ja toteutusvaihetta vastaa testausvaihe. Kun käsittelyssä on jokin suunnittelu- ja toteutusvaiheen osatehtävä, samaan aikaan määritellään vastaavan testausvaiheen testejä.
- Koska testit määritellään samaan aikaan suunnittelun ja toteutuksen kanssa, testaus on ajan tasalla. Testaus siirretään viimeisestä osavaiheesta *sateenvarjotoiminnoksi* (umbrella activity).
  - Sateenvarjotoiminto on sellainen tehtävä, joka on läsnä koko projektin kestoajan. Esimerkiksi dokumentointi on sateenvarjotoiminto.

V-mallin vaihejako

- V-mallin vaihejako vastaa perinteistä vesiputousmallin vaihejakoa:
  - Vaatimusten kartoitus -> Hyväksymistestit
  - Vaatimusten analyysi -> Järjestelmätestit
  - Arkkitehtuurisuunnittelu -> Integrointitestit
  - Komponenttisuunnittelu -> Komponenttitestit
  - Koodaus ja yksikkötestaus
  - Komponenttitestaus
  - Integrointitestaus
  - Järjestelmätestaus
  - Hyväksymistestaus

V-malli kaaviokuvana

- Yleensä V-malli kuvataan V-kirjaimena (siitä mallin nimi), jossa vasemmassa haarassa ovat vesiputousmallin perustehtävät ja oikeassa haarassa ovat testaustehtävät:



V-malli ja dokumentaatio

- V-malli perustuu vesiputousmallin tapaan huolelliseen dokumentaatioon.
- Ehkä eniten V-malli eroaa vesiputousmallista dokumenttien tarkastuksissa. Siinä missä vesiputousmallissa dokumenttien tarkastukset ovat yksi tekniikka muiden joukossa, V-mallissa ne ovat tärkeä perustekniikka.
- Puhtaimmillaan V-mallissa tarkastetaan sekä jokaisen osavaiheen tuotos että vastaavan osavaiheen testitapausmäärittely.

V-mallin edut

- Koska V-malli on läheistä sukua vesiputousmallille, kaikki vesiputousmallin edut ovat yhtä lailla V-mallin etuja. Näiden lisäksi:
  - Testaus on hyvin hallittu sateenvarjotoiminto, eli sitä tehdään koko projektin ajan.
  - Aikataulun tai budjetin ylittyessä tuote ei jää täysin testaamatta.
  - Hyvä testausdokumentaatio mahdollistaa toimivan *regressiotestauksen* (regression testing) eli vanhojen testitapausten suorituksen uudestaan.

V-mallin haitat

- V-malli kärsii samoista ongelmista kuin vesiputousmalli:
  - Vaatimukset eivät saa muuttua!
    - Jos tämä vaatimus ei toteudu, vesiputousmallia ei kannata käyttää.
  - Asiakas näkee valmista vasta projektin päättyessä.
  - Projektityöntekijät saattavat turhautua jatkuvaan dokumentointiin ja tarkastuksiin.
  - Malli ei suosi luovuutta (vaikka suosii vesiputousmallia enemmän)
  - Väärille urille joutuneen projektin saattaminen oikeaan suuntaan on vaikeaa.
  - Myöhässä olevaa projektia on vaikeaa saada takaisin aikatauluun.
- Edellisten vesiputousmallin ongelmien lisäksi puhtaan V-mallin mukaiset projektit saattavat olla kalliita. Mallissa tehdään jopa vesiputousmallia enemmän dokumentaatiota, minkä lisäksi kaikki dokumentit pitäisi tarkastaa ennen hyväksymistä.

Prototyyppimalli

- Ensimmäinen muuttuvien vaatimusten ongelmaan ratkaisua etsinyt prosessimalli on *prototyyppimalli* (prototyping approach). Siinä tuotteesta tehdään toteutuksia, joissa ulkoiset liittymät ovat kunnossa mutta sisäinen logiikka on vajaa.
- Prototyyppimalli ei oikeastaan ole malli vaan malliperhe. Kirjallisuudessa on varsinkin 80-luvulla esitelty joukko muuttuviin vaatimuksia tukevia prosessimalleja, joissa asiakas antaa palautetta vajaasta tuotteesta: prototyypistä.
- Prototyyppeihin perustuvat prosessit elävät ja voivat hyvin, vaikka tällä hetkellä tehdään enemmän iteratiivista ohjelmistokehitystä.

Prototyypit

- Prototyyppi on tuotteen osittainen implementaatio, joka esitetään joko loogisena mallina (paperiprototyyppinä) tai suoritettavana ohjelmistona
- Loppukäyttäjät kokeilevat prototyyppiä ja antavat siitä palautetta kehitystiimille.
- Prototyyppi ei ole lopullinen ohjelmisto. Varsinainen ohjelmiston kehitystyö alkaa, kun loppukäyttäjät ovat tyytyväiset prototyyppiin.
- Lopullinen kehitystyö voidaan tehdä minkä tahansa prosessimallin mukaisesti. Prototyyppimalli on siten enemmän apuväline kuin varsinainen prosessimalli.

Prototyyppimallien työvaiheet

- Prototyyppimallissa on yleensä seuraavat työvaiheet (Galin, 2003):
  1. Vaatimusten keruu ja analyysi
  2. Lyhyt suunnitteluvaihe
  3. Prototyypin rakennus
  4. Asiakkaan palaute prototyypistä
  5. Suunnittelun ja prototyypin päivitys
  6. Jos asiakas ei ole tyytyväinen protoon, palataan askeleeseen 4
  7. Jos asiakas on tyytyväinen protoon, varsinainen kehitystyö aloitetaan

Prototyyppien käyttö

- Prototyyppien käyttö toimii parhaiten pienillä ohjelmistoilla tai osajärjestelmillä. Kokonaisen suuren tai keskisuuren järjestelmän prototyyppien teko on hankalaa.
- Prototyyppimallin ehdoton vaatimus on, että prototyyppien teko on helppoa. Käytännössä tämä tarkoittaa, että prototyyppejä tehdessä on voitava käyttää mahdollisimman paljon uudelleenkäytettäviä komponentteja. Sopiva sovelluskehitin auttaa samoin prototyyppien teossa.

Prototyyppien toteutus

- Prototyypeissä kannatta keskittyä ulkoisiin rajapintoihin: siis käyttöliittymiin. Sisäinen toteutus tehdään mahdollisimman helpolla.
  - Jos prototyyppi on ns. *poisheitettävä prototyyppi* (throwaway prototype), joka aloitetaan aina tyhjästä, niin sen sisäinen toteutus kannattaa tehdä mahdollisimman vähällä vaivalla.
  - Jos prototyyppi on ns. *jatkokehitettävä prototyyppi* (evolutionaly prototype), joka perustuu edellisiin prototyyppeihin, niin sen sisäiseen toteutukseen kannattaa käyttää vähän enemmän vaivaa.
  - Poisheitettävät prototyypit ovat yleisempiä kuin jatkokehitettävät. Jatkokehitettäviä tarvitaan, jos prototyypin tekoon vaadittava työmäärä kasvaa suuremmaksi kuin jo tehdyn koodin päivittäminen.

Käyttöliittymäpohjaiset prosessimallit

- Prototyyppimallin erikoistapaus ovat *käyttöliittymäpohjaiset prosessimallit* (user interface driven process models). Niiden lähtökohta on, että ohjelmiston suunnittelun pitää lähteä toiminnallisuuden sijaan käyttöliittymästä.
- Käyttöliittymäpohjaisessa prosessimalllissa asiakkaalle tehdään ensin yksi tai useampi käyttöliittymämalli. Asiakas kokeilee malleja ja kertoo, miten hyvänä hän pitää mallien käytettävyyttä.

Käyttöliittymäpohjaiset prosessimallit 2

- Käyttöliittymämallien pohjana ovat asiakkaan listaamat tarpeet, eivät palvelut. Vasta tarpeiden ja käyttöliittymän selvittämisen jälkeen selvitetään muut vaatimukset.
- Ensimmäiset käyttöliittymämallit voivat olla kuvia. Myöhemmissä voi olla käyttöliittymään sidottua toiminnallisuutta.
- Käyttöliitymämallit ovat prototyyppejä, joten käyttöliittymäpohjaiset prosessimallit ovat prototyyppimallin erikoistapauksia.

Prototyyppimallin edut

- Prototyyppien teko sopii mihin tahansa prosessimalliin.
- Asiakas näkee valmiin tuotteen mallin hyvin aikaisessa vaiheessa projektia.
- Prototyypit antavat asiakkaalle pelkkää tekstidokumentaatiota selkeämmän kuvan ohjelmiston toiminnasta.
- Prototyypit selventävät rajapintojen käyttöä.
- Prototyyppien avulla ongelmakentästä saattaa löytyä käsittelemättömiä alueita.

Prototyyppimallin haitat

- Prototyyppien teko on projektin lisäkustannus.
- Prototyyppien teko vaatii hyvät työkalut ja paljon uudelleenkäytettävää koodia.
- Malli ei sovi isoihin projekteihin.
- Kustannusarvion ja aikataulun teko vaikeutuu.
- Asiakas ei aina ymmärrä, miksi valmiilta vaikuttava prototyyppi ei ole lopullinen ohjelmisto.
  - Miksi proto ei ole luovutettava tuote? Koska prototyypeissä oiotaan toteutuksessa. Jopa pitkänkin evoluution läpikäynyt prototyyppi on prototyyppi, joka ei täytä lopulliselle tuotteelta vaadittavaa laatua.
- Joskus on vaikea sanoa, milloin on tehty riittävästi protoja ja on aika ruveta varsinaiseen kehitystyöhön.

Formaali ohjelmistokehitys

- Tavalliset ohjelmistoprosessit ovat kärjistäen yritys-erehdys-prosesseja. Niissä tehdään jotain, tarkastetaan tuotos ja tarvittaessa päivitetään tulosta.
- Muissa insinööritieteissä toimitaan harvoin tällä tavalla. Niissä tuotanto perustuu *matemaattiseen mallinnukseen* (mathematical modelling).
  - Esimerkiksi sillanrakennus perustuu tarkkoihin lujuuslaskelmiin. Sillan on oltava kerralla oikein. Sitä ei tuosta vaan ”päivitetä”.

Formaalien menetelmien taustaa

- Myös ohjelmistoprosesseissa voidaan käyttää matemaattista mallinnusta. Tällaisia prosesseja kutsutaan *formaaleiksi prosesseiksi* (formal processes) ja niissä käytettyjä menetelmiä *formaaleiksi menetelmiksi* (formal methods).
- Formaaleissa menetelmissä lähdetään siitä, että matemaattisen formalismin avulla projektista saatu tuotos todistetaan oikeaksi.
- Pisimmälle vietynä formalismissa tehty ohjelmisto todistetaan saatu ohjelmisto oikeaksi. Tämä on vähintään hyvin kallista ja usein jopa mahdotonta. Menetelmää käytetään, kun on oltava aivan varmoja, että ohjelmisto toimii oikein.

Kevyemmät formaalit menetelmät

- Ohjelman oikeaksi todistamista keveämmät formaalit menetelmät sopivat paremmin moderniin ohjelmistokehitykseen.
- Yksinkertaisimillaan matemaattista formalismia voidaan käyttää kuvaukseen formaalista mallista ohjelmiston osaan.
  - Esimerkiksi relaatiotietokantojen SQL-kyselykieli perustuu matemaattisesti hyvin hallittuun relaatioalgebraan. Näin relaatioalgebraa voidaan käyttää hyväksi SQL-kyselyiden toteuttamisessa.
- *Formaali spesifiointi* (formal specification) sopii tilanteisiin, joissa halutaan varmistaa ohjelman spesifikaation ristiriidattomuus.
  - Ensin vaatimukset kuvataan sopivalla formaalilla kielellä
  - Tämän jälkeen tarkastetaan automaattisesti, että kuvauksessa ei ole ristiriitoja.
- *Formaalin verifioinnin* (formal verification) avulla todistetaan, että verifioitava algoritmi toimii oikein.

Cleanroom

- Tunnetuin formaaleihin menetelmiin perustuva prosessimalli on *Cleanroom* (Linger, 1993). Se on iteratiivinen prosessimalli, jossa formaaleja menetelmiä käytetään spesifioinnissa ja suunnittelussa.
- Mallin ideana on virheiden korjaamisen sijaan virheiden välttäminen. Tähän päästään huolellisella formaalilla suunnittelulla ja ohjelman oikeaksi todistamisella.

Cleanroom

- Cleanroomin erikoisuus on yksikkötestauksen puuttuminen.
  - Cleanroom-prosessimallissa yksikkötestauksen korvaa formaali suunnittelu. Ohjelma jaetaan sisäkkäisiin *laatikoihin* (box), joilla on yksi alku- ja loppupiste. Kukin laatikko todistetaan oikeaksi.
  - Laatikot esitellään kolmella abstraktiotasolla. Korkeimmalla tasolla laatikosta tiedetään sen syöte ja tulos. Seuraavalla tasolla tiedetään laatikon syötteestä ja tapahtumista johtuvat tilat. Kolmannella tasolla tiedetään ohjelmakoodi. Oikeaksi todistaminen tehdään korkeammasta abstraktiosta matalampaan.

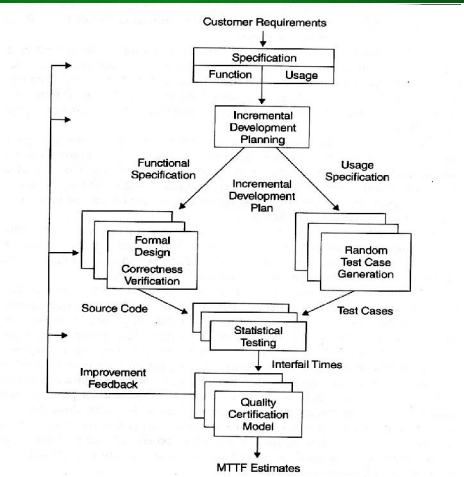
Cleanroom 2

- Cleanroomissa ei tehdä tavallista järjestelmätestausta. Sen sijaan mallissa tehdään *tilastollista testausta* (statistical testing). Testitapaukset generoidaan mahdollisuuksien mukaan automaattisesti sen mukaan, mitä osia ohjelmistosta käytetään eniten.
- Cleanroomilla on saatu erittäin laadukkaita ohjelmistoja. Linger raportoi *kaikkien* löydettyjen virheiden lukumääräksi keskimäärin 2,3 virhettä / 1000 koodiriviä. Normaalissa ohjelmistokehityksessä luku on noin 50. Jopa täysin virheettömiä tuloksia on raportoitu. Toisin sanoen suunnittelun jälkeen koodi on mennyt suoraan läpi kääntäjästä eikä testauksessa ole löytynyt ainuttakaan virhettä!
- Cleanroom sopii parhaiten sellaisiin pieniin ja keskisuuriin projekteihin, joiden tuotosten on oltava erittäin luotettavia. Menetelmää voidaan käyttää isoihin projekteihin, jos projekti on ositettavissa sopivan kokoisiin osaprojekteihin ja iteraatioihin.

Cleanroom-prosessin vaiheet

- Spesifiointi (specification)
  - Tuloksena saadaan kaksi speksiä: toiminnallinen spesifikaatio ja käyttöspesifikaatio.
    - Toiminnallinen spesifikaatio määrittelee järjestelmältä vaadittavan ulos näkyvän toiminnallisuuden kaikissa mahdollisissa tilanteissa.
    - Käyttöspesifikaatio määrittelee järjestelmän käyttötavat ja tietyn käyttötavan (käyttötapauksen) tapahtumisen todennäköisyyden.
- Syklien suunnittelu (increment planning)
  - Tiimi tekee alustavan suunnitelman siitä, miten tuotteen toteutus jaetaan iteraatioihin.
- Suunnittelu ja verifiointi (design and verification)
  - Cleanroomissa on kaksi tiimiä. Toinen suunnittelee ja toteuttaa koodin. Toinen generoi testitapauksia tilastollista testausta varten.
- Laadun sertifiointi (quality sertification)
  - Kehitystiimi integroi kirjoittamansa koodin aiemmin integroituun koodiin, minkä jälkeen testaustiimi tekee koko koodille tilastollisen testauksen.

Cleanroom-prosessi



Kuvalla (c) Richard Linger 1993

Formaalien menetelmien edut

- Korkeatasoinen tuotteen laatu.
  - Tosin: edelleen kiistellään siitä, saavutetaanko formaaleilla menetelmillä parempilaatuisia ohjelmistoja kuin tavallisilla menetelmillä.
- Käytössä on matematiikan koko ilmaisuvoima.
- Puhutun kielen moniselitteisyyttä ei esiinny.
- Parhaimmillaan tuote voidaan verifioida automaattisesti.

Formaalien menetelmien haitat

- Sopii vain sellaisiin projekteihin, joiden tuotokset voidaan mallintaa matemaattisesti.
- Useimmilla työntekijöillä ei ole riittäviä matemaattisia valmiuksia.
- Oikeaksi todistaminen on kallista.
- Dokumentaation lukeminen vaatii formaalien menetelmien asiantuntijan.
  - Toisin sanoen projektista raportointi asiakkaalle, laadunvalvontaryhmälle ja omille johtajille on hyvin hankalaa.