TNM094 – Medietekniskt kandidatprojekt

Testning

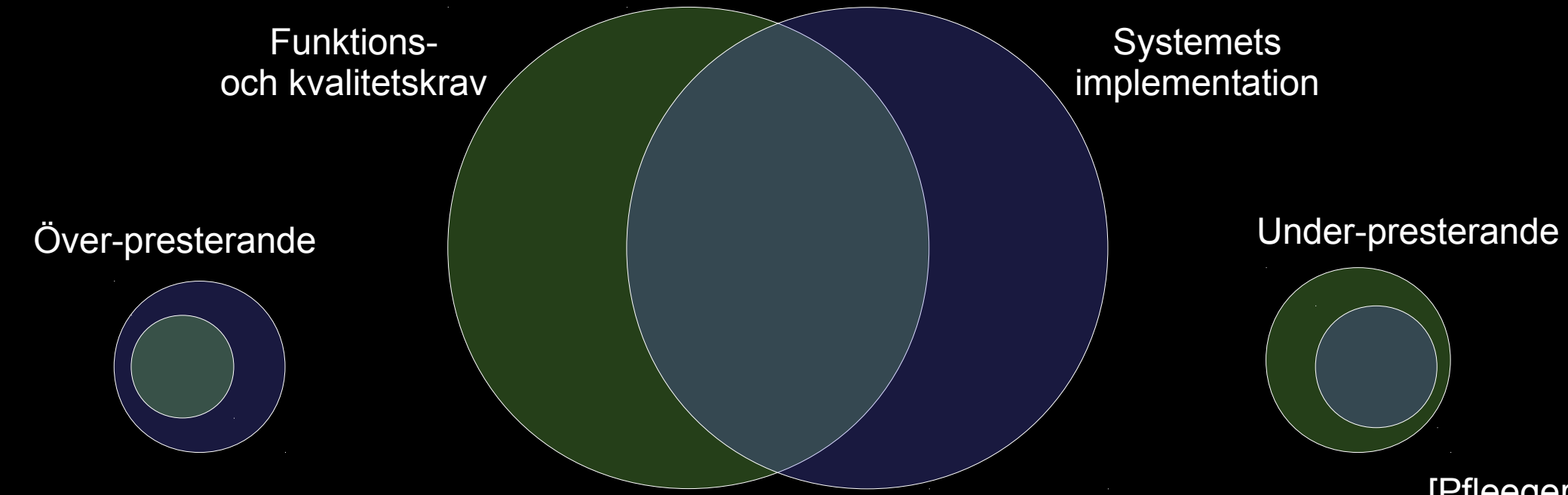
Brister och körfel

- Du är inte färdig när du har implementerat!
- Terminologi
  - körfel, failure
    - vilket inkorrekt beteende som helst
    - t ex släppa känslig information till icke-auktoriserade
    - t ex krascha när en användare gör någonting oväntat
  - brist, programfel, fault
    - det i systemet som leder till körfel
    - t ex felaktig implementation av specificerade krav
    - t ex defekt design
    - t ex användning av borttaget objekt

[Pfleeger]

Testning

- Syfte
  - *inte* att testa om programmet kan köras
  - verifiera att implementationen matchar kraven
  - räkna med att brister finns – målet är att hitta dem
  - QA – quality assurance, vi siktar på en viss kvalitetsnivå



[Pfleeger]

Testning

- Identifiera fel
  - identifiera och analysera felet
  - korrigera inte utan hitta orsaken
- Korrigera fel
  - först efter noggrann analys vet vi orsaken
  - ta bort hela bristen, fixa inte bara raden där körfelet uppstod (krav, arkitektur, programdesign, implementation)
- QA handlar om att ha processer och rutiner
  - För att identifiera fel och för att korrigera fel

[Pfleeger]

Olika sorters fel

- Konsekventa fel
  - Algoritmiska fel
  - Beräknings-/precisions-fel
  - Dokumentationsfel
- Transienta (övergående) fel
  - Stress, belastning, kapacitet
  - Timing och koordination
- Miljö-associerade fel
  - Fel maskinvara
  - Fel-implementerad standard

[Pfleeger]

Varifrån kommer felen?

- Utifrån
  - t ex i kommunikationen vid elicitationen av krav
  - t ex i tolkningen av körmiljö
- I designprocessen
  - t ex vid feltolkning av algoritm- eller design-egenskaper
  - t ex trött designer missar en aktivitet i modelleringen
- Vid implementation
  - t ex klippa-klistra-fel, felstavning, feltolkning, fel scope
  - t ex misstolkad språkkonstruktion eller API
- Vid dokumentation
  - t ex missad uppdatering av dokumentation vid förändringar
  - t ex misstolkning av funktionens beteende

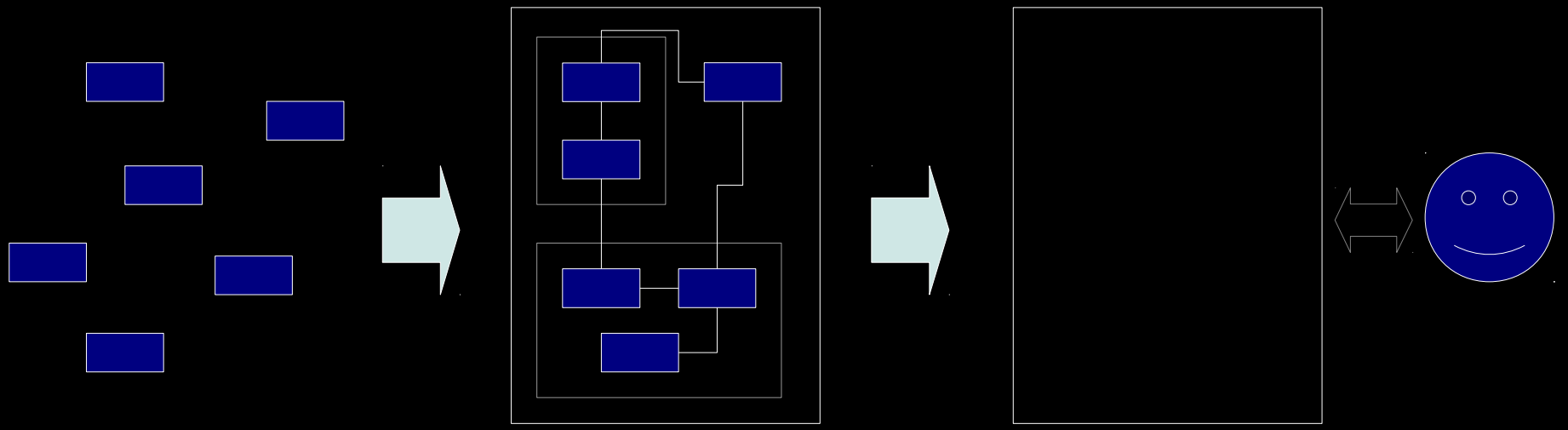
[Pfleeger]

Sociala aspekter

- Känslor
  - det är lätt att ta kritik personligt
  - en del är obekväma med att andra granskar deras kod
  - sårade känslor och krossade egon har ingen plats i utvecklingsprocessen
  - recensioner är givande både i processen och personligen
- Egoless programmering
  - alla äger allt
  - komponenter är en del av systemet, inte ägs av författaren
- Oberoende test team
  - känslomässigt fristående
  - mer specialiserad kompetens
  - inte tänker på samma linjer som orsakade problemet

[Pfleeger]

Test-faser



Enhetstest
(unit test)

Integrationstest

System-test:

Funktions-test
Prestanda-test
Acceptans-test
Installations-test

[Pfleeger]

Enhetstest

- Syfte
  - Primärt: funktionstestar en modul, klass, metod, etc
  - (Även: kodkvalitet)
- För
  - korrekt beteende (mot krav)
  - typiskt fokus på algoritmiska fel
  - även pålitlighet och robusthet

[Pfleeger]

Tekniker

- Kodgranskning
  - Parprogrammering
  - Partner (four eye) review
  - Walk-through
  - Code inspection
- Automatiska körtest
- Automatisk statisk analys
- (Formell analys och bevis)

[Pfleeger]

Individuell granskning

- Parprogrammering
  - Ögonblicklig och kontinuerlig ”granskning”
  - Ögonblicklig och kontinuerlig justering
- Partner-granskning (four-eye review)
  - koden granskas av en eller två kolleger
  - före merge med trunk/master
  - via epost, pull request, dedicerade system, etc
- Fix
  - informell justering med ny granskning
  - formella protokoll, retur och om-programmering

[Pfleeger]

Grupp-granskning

- Walk-through
  - Presentation inför ett review team
    - kod och dokumentation
    - fokus på att identifiera problem, inte fixa dem
    - informellt – fokus på koden, inte programmeraren
- Code Inspection
  - Mer formell än walk-through
    - använd lista över problemområden
    - domänspecifik lista
    - från tidigare utvecklingsfaser
    - på annat sätt identifierade problemområden
  - Steg
    - inledande möte – övergripande genomgång av koden; bestäm mål
    - inspektion – individuellt arbete; läs och dokumentera
    - avslutande möte – identifiera falska positiva eller ytterligare fel

[Pfleeger]

Viktiga aspekter

- Kodgranskning hittar många
  - programmeringsfel
  - läsbarhetsproblem
  - designfel på låg nivå
- Ju tidigare ett problem hittas, desto bättre
  - felets ursprung är tydligare
  - mindre kod beror på den felaktiga koden
  - arbetet är fortfarande färskt i minnet
- Kan dock inte automatiseras

[Pfleeger]

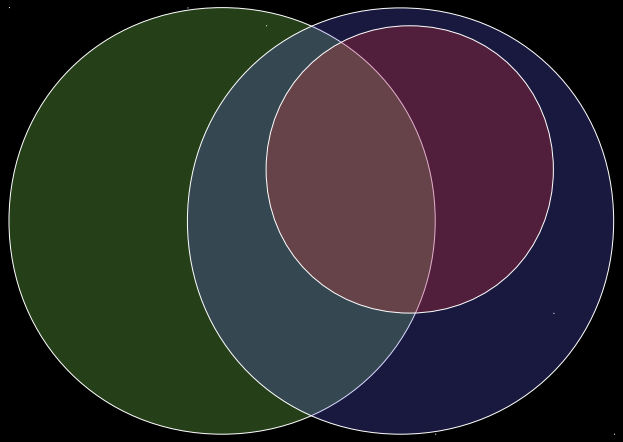
Automatiska körtest

- Kör enheten och kontrollera resultatet
  - (kallas ibland *regressionstest*)
- Input
  - skapa exempelkörning med bestämda tillstånd och input
- Kontrollera resultatet (post conditions)
  - returnerat värde
  - olika typer av undantag (exception)
  - interna tillstånd
  - sparade filer
  - genererade bilder
  - (etc …)

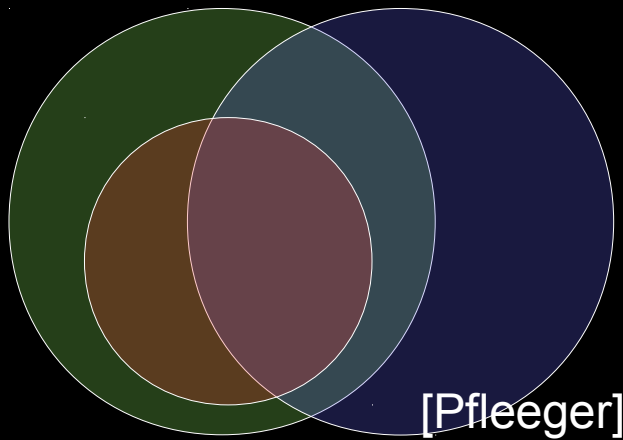
[Pfleeger]

Test-fall

- Ett test består av en uppsättning test-fall
  - open-/clear-/whitebox-testning
    - Anpassa fallen utifrån implementationen
    - Strävar mot full kodtäckning
    - Testar inte vad enheten *borde* göra



  - closed-/blackbox-testning
    - Anpassa fallen utifrån dokumentationen eller kraven
    - Arbeta med input-klasser
    - Riskerar att sakna viktiga input-klasser



[Pfleeger]

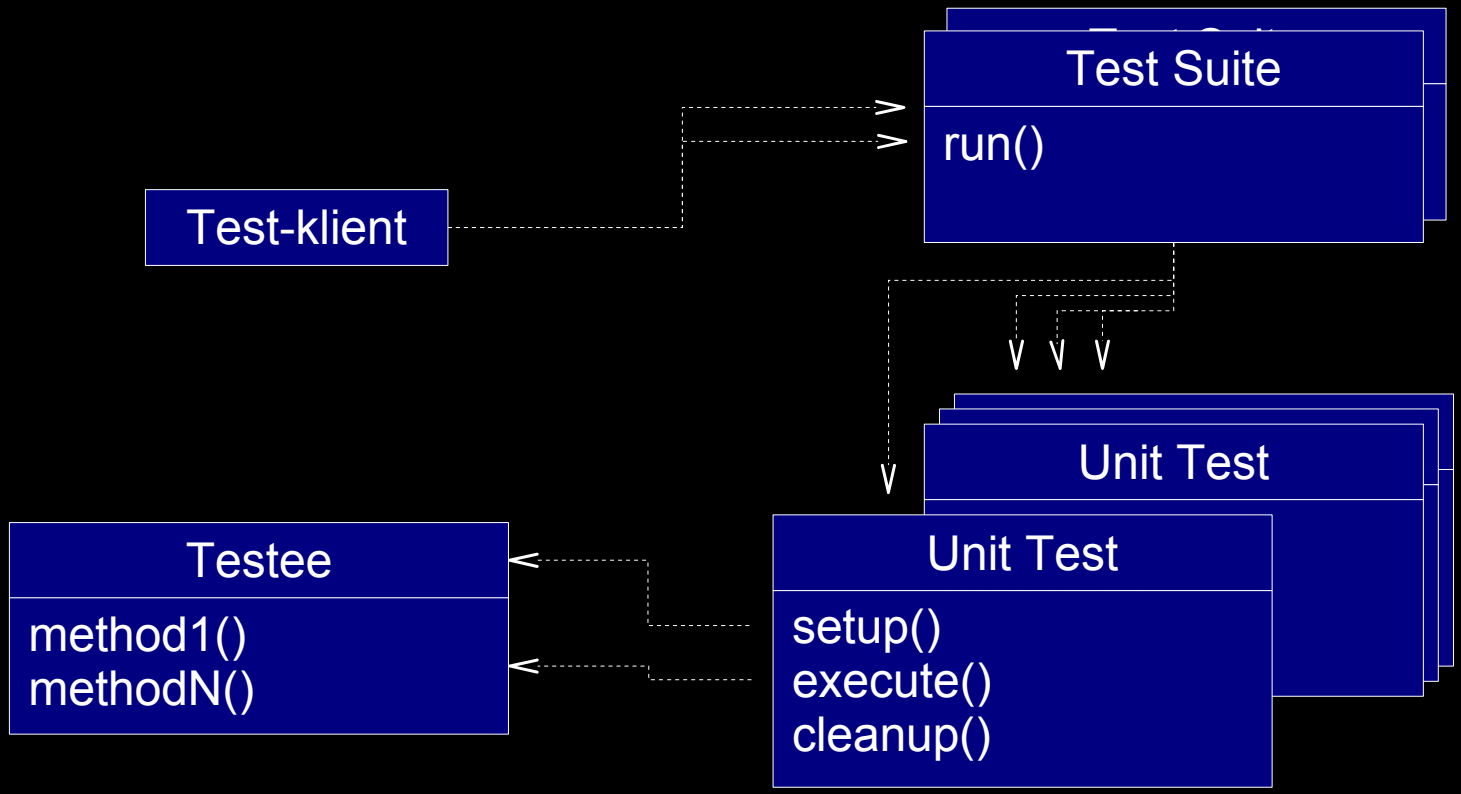
Kodtäckning

- Kodtäckning
  - Mått på hur mycket av koden som testas
  - Alla test-fall bör tillsammans täcka 100% av koden
- Säger inte om testen är rätt
  - Det som räknas ut kan vara fel
  - Interna tillstånd kan sättas till fel värde

[Pfleeger]

Kör-test

- Manuell eller automatisk körning av test
  - Test-klient kör en test-samling som kör test, ett i taget
  - CppUnit, JUnit, etc



[Pfleeger]

Verktyg för testning

- Statisk programanalys
- Dynamisk programanalys
- (Automatisk test-case-generator)
- (Stubbs- och driver-generator)

[Pfleeger]

Statisk programanalys

- Analyserar själva programkoden, inte körning
  - Kontrollerar alla delar och förgreningar
  - Letar efter direkta fel
  - Identifierar även problematisk struktur
- Kan även arbeta mot binärkod
- Rekommenderas av vissa organisationer
  - Motor Industry Software Reliability Association (MISTRA)
  - Office for Nuclear Regulation (ONR)
  - Food and Drug Administration (FDA)

[Pfleeger]

Dynamisk programanalys

- Kontrollerar vad som händer under körning
  - Felaktig minnesanvändning
  - Access till osynkroniserat minne från olika trådar
- Statistik
  - Antal anrop till en funktion
  - Tid spenderad i en funktion
- Interaktiv analys
  - Pausa programmet på vissa rader (breakpoints)
  - Läsa av variabler och anropsstack
  - Övervaka uttryck för förändring

Stopp-kriterier vid testning

- När slutar man att leta efter fel?
  - det senast hittade felet är aldrig det sista
  - använd kvalitetskrav för att bestämma stopp-kriterier
- Stopp-kriterier
  - Antal fel per kLoC (1000 rader kod)
  - Sannolikhet att systemet är felfritt
- (J m f Agil utveckling)
  - Test Driven Development – kör-testa allt
  - Tidsbegränsad testning

[Pfleeger]

Hitta fel-benägna enheter

- Syfte
  - hitta enheter som troligen innehåller fel
  - testa dessa först för att undvika följdfel
  - allokera extra resurser till att testa dessa
- Tekniker
  - kända fel-benägna funktionsområden / ansvar
  - spåra antalet fel som hittats under utvecklingsarbetet
  - beräkna komplexitet utifrån mätvärden
  - (→ Classification tree analysis)

[Pfleeger]

Integrationstest

- Testa kombinationer av moduler
  - Kommunikation, protokoll, anrop
  - Schemalagd integration
    - Testa integration vid speciella tillfällen (t ex efter sprint)
    - Testa integration mellan vissa moduler i taget
    - Uses graphs – beroende-information
    - Stubs – kod som simulerar en enhet som anropas
    - Drivers – kod som simulerar en enhet som anropar

[Pfleeger]

Kontinuerlig integration

- Integrera varje förändring direkt i systemet
  - kodgranskning säkerställer design- och kod-kvalitet
  - körtest säkerställer att varje del alltid fungerar (regressionstest)
  - systemtest säkerställer att integrationen lyckats
- Viktiga verktyg
  - CI-server – testar alla relevanta konfigurationer
- Principer
  - Enklare – slipper ”stub” och ”driver”
  - Fler platser att leta efter fel på

Systemtestning

- Testa systemet som helhet
  - Ofta mer komplicerat än enhetstest
  - Funktions-test – finns funktionerna?
  - Prestanda-test – finns prestandan?
  - Acceptans-test – vad säger kunden?
  - Installations-test – kan vi leverera?

[Pfleeger]

Funktionstest

- Syfte
  - grundläggande test av systemets funktionalitet
- Teknik
  - Thread testing
    - testa en serie med handlingar på en funktion
    - steg-för-steg-instruktioner och observerbara resultat
    - ska inkludera även felaktig input vid behov
  - (Cause-and-effect graphs)

[Pfleeger]

Prestandatest

- Syfte
  - testa systemets prestanda

- Stress-/volym-tests – stressa systemet över gränserna under kort tid
- Konfigurations-test – analysera olika program- och maskinvarukonfigurationer
- Kompatibilitets-test – testa på och mot olika system
- Säkerhets-test – testa tillgänglighet, integritet och konfidentialitet (CIA)
- Timing-test – testa responstid
- Kvalitets-test – utvärdera pålitlighet, robusthet, underhållbarhet, etc
- Dokumentation-test – utvärdera guider, teknisk dokumentation, etc
- Mänskliga faktorer – utvärdera användbarhet

[Pfleeger]

Acceptanstest

- Syfte
  - kontrollera att kunden anser att systemet uppfyller kraven
- Styrs av kunden eller en kundrepresentant
- Typer
  - Benchmark-test
    - praktiska test som definierats av kunden
    - körs under typiska förhållanden för systemet
  - Pilot-test
    - Installera och använda systemet
    - Alfa-test – hos utvecklarna
    - Beta-test – kunden kör piloten själva
  - Parallell-testning
    - det nya systemet körs parallellt med det gamla

[Pfleeger]

Testplan

- Syfte
  - organisera test-aktiviteter
  - schemaläggning utifrån deadlines
  - visa för kunden hur vi demonstrerar
    - att systemet uppfyller funktionskraven
    - att systemet uppfyller kvalitetskrav
  - att skriva testplan tvingar oss att förstå
    - systemets mål
    - systemets design

[Pfleeger]

Testplan

- Organisation
  - vem är det som testar
  - schema och procedurer
  - rapporter och dokumentation
- Syfte med testningen
  - man kan aldrig garantera ett fel-fritt system
  - hur hög kvalitet siktar vi på?
- Hur testen genomförs
  - vilka test och hur
  - verktyg, APIer, automation, etc
  - stopp-kriterier

[Pfleeger]

Testrapport

- Format och innehåll
  - beror mycket på process och företag
  - blir en del av QA
- Felrapport
  - hur uppstår problemet, hur går det att återupprepa (förutsättningar, timing, data, mekanism, etc)
  - referenser till berörda delar av systemet och eventuella krav (inklusive gren, revision, etc)
  - nivå (severity) och kostnadsbedömning
- Lösning
  - mekanism, orsak
  - ansvarig utvecklare, tidsplan
  - diskussioner, mötesprotokoll
  - slutlig avrapportering av tid, kostnad och genomförda förändringar

[Pfleeger]

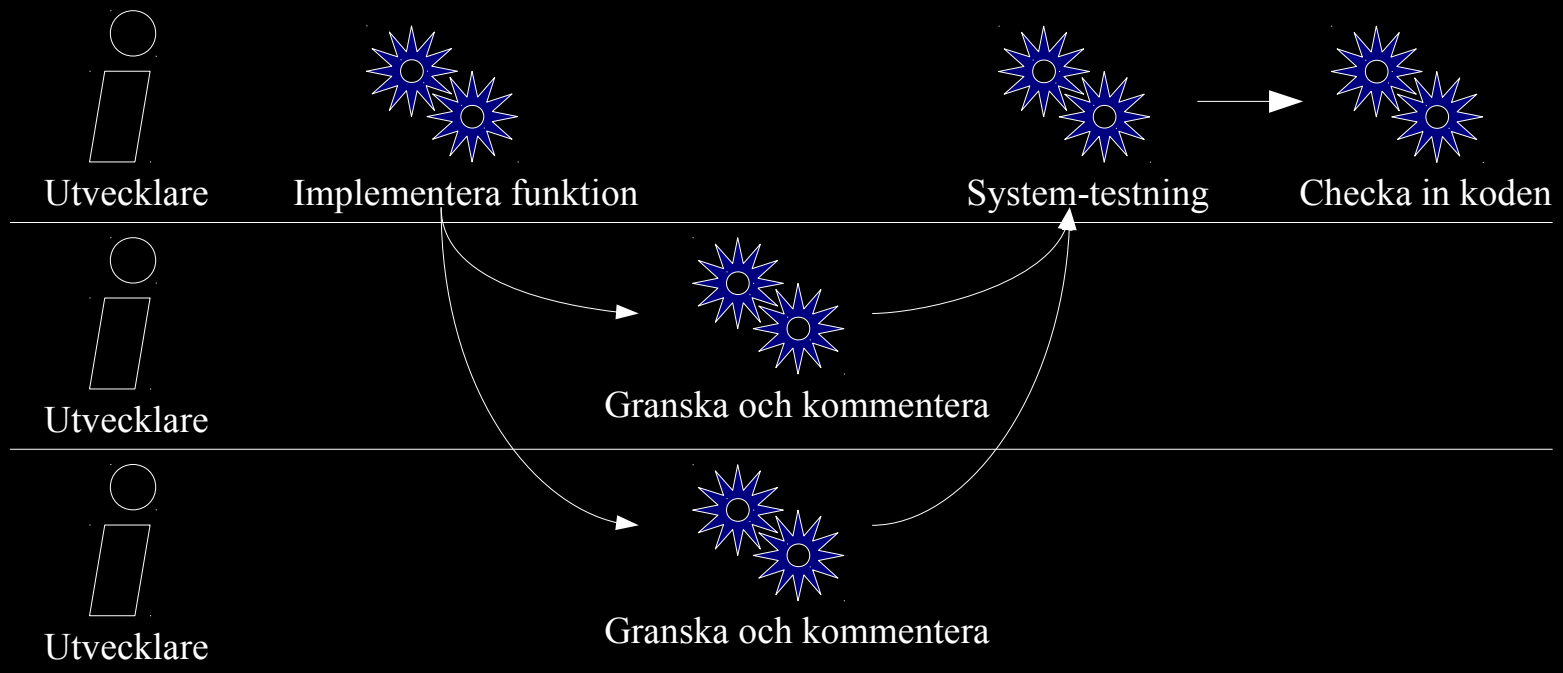
Testning i Agil utveckling

- Testning är en del av utvecklingsarbetet
  - görs för varje uppgift, post, iteration
- Kontinuerlig integration
  - för att minimera arbetet med integration
- Test-driven utveckling (TDD)
  - kraven specificeras som exempel
  - acceptanstest definieras från exemplen
  - körtest definieras från kraven
  - alla test skapas innan implementationen som ska testas

[Pfleeger]

Test-processer (exempel 1)

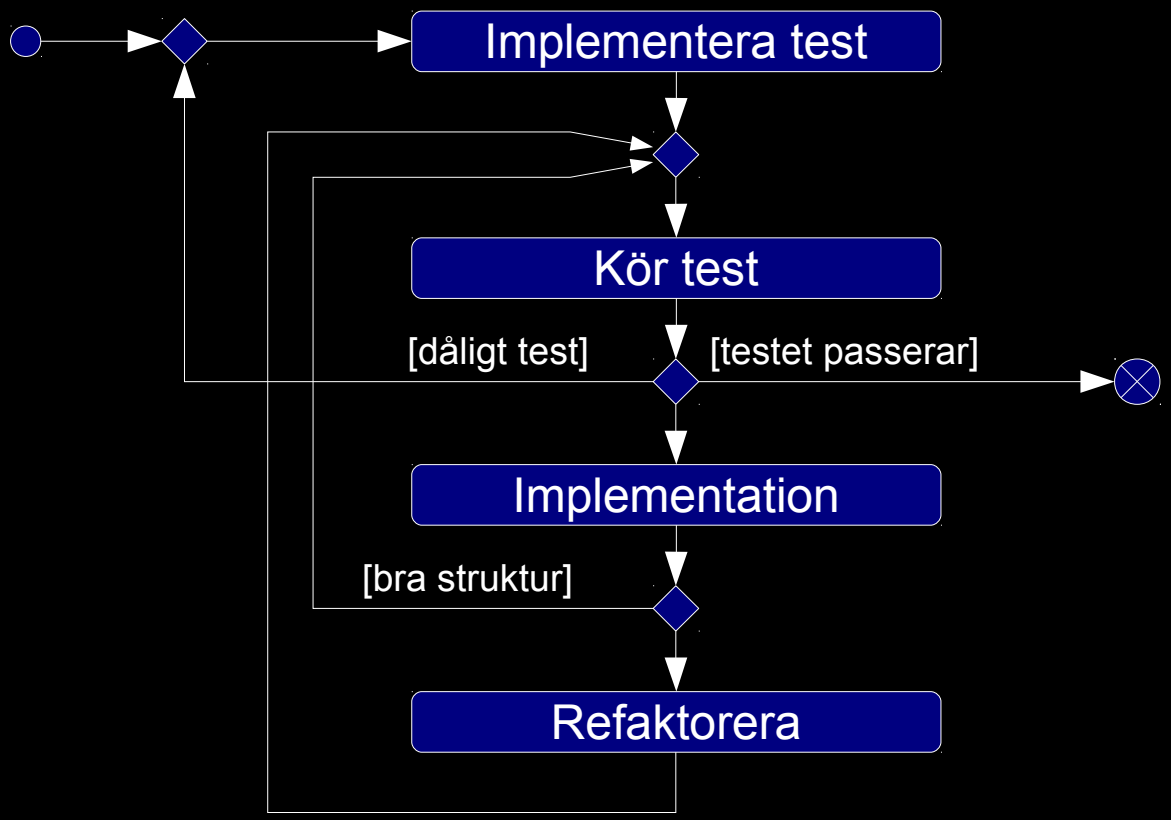
- Partner-granskning och system-testning



(RUP work-flow diagram)

Test-processer (exempel 2)

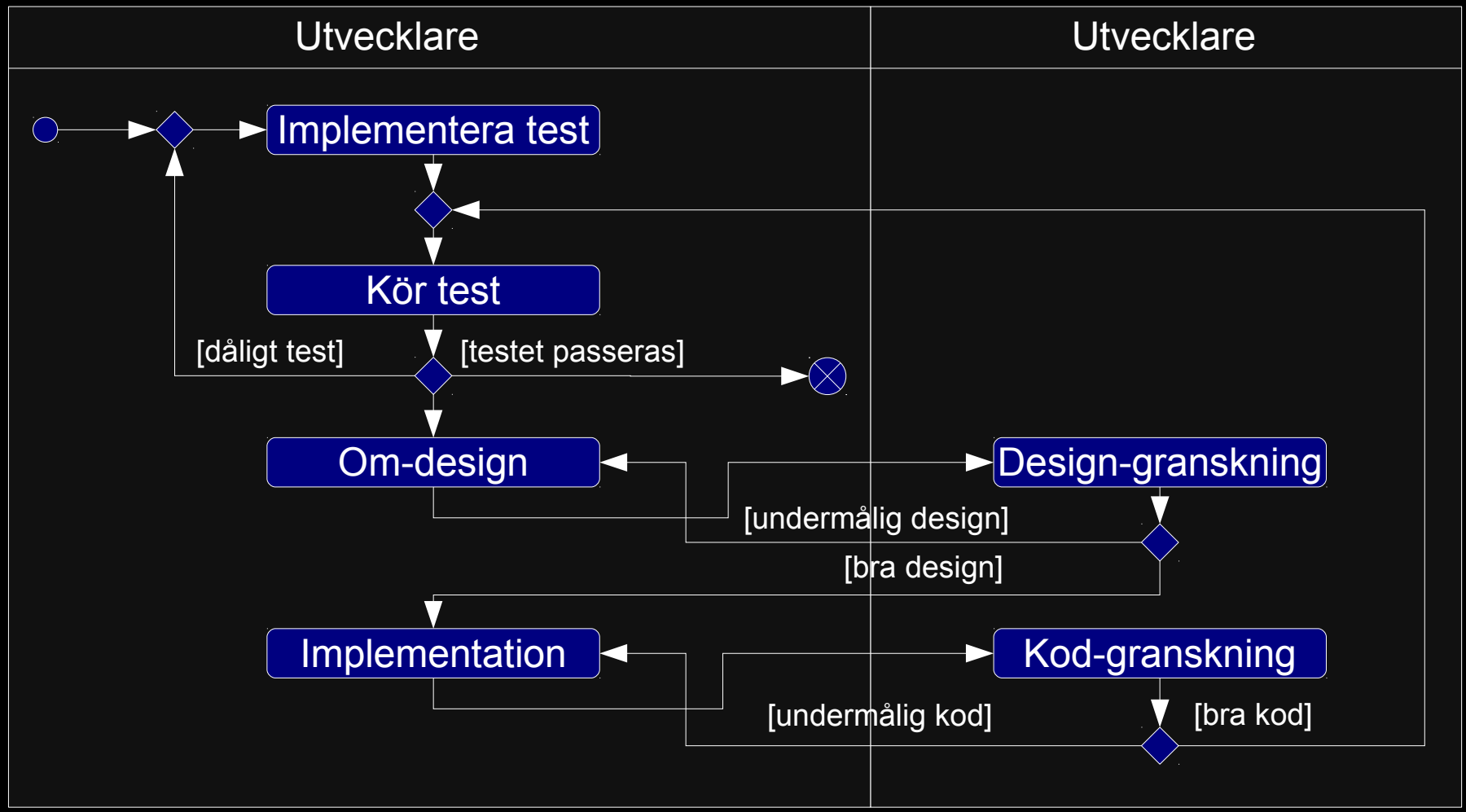
- Test-driven utveckling (TDD)
  - För varje post eller uppgift



(UML2 Aktivitets-diagram)

Test-processer (exempel 3)

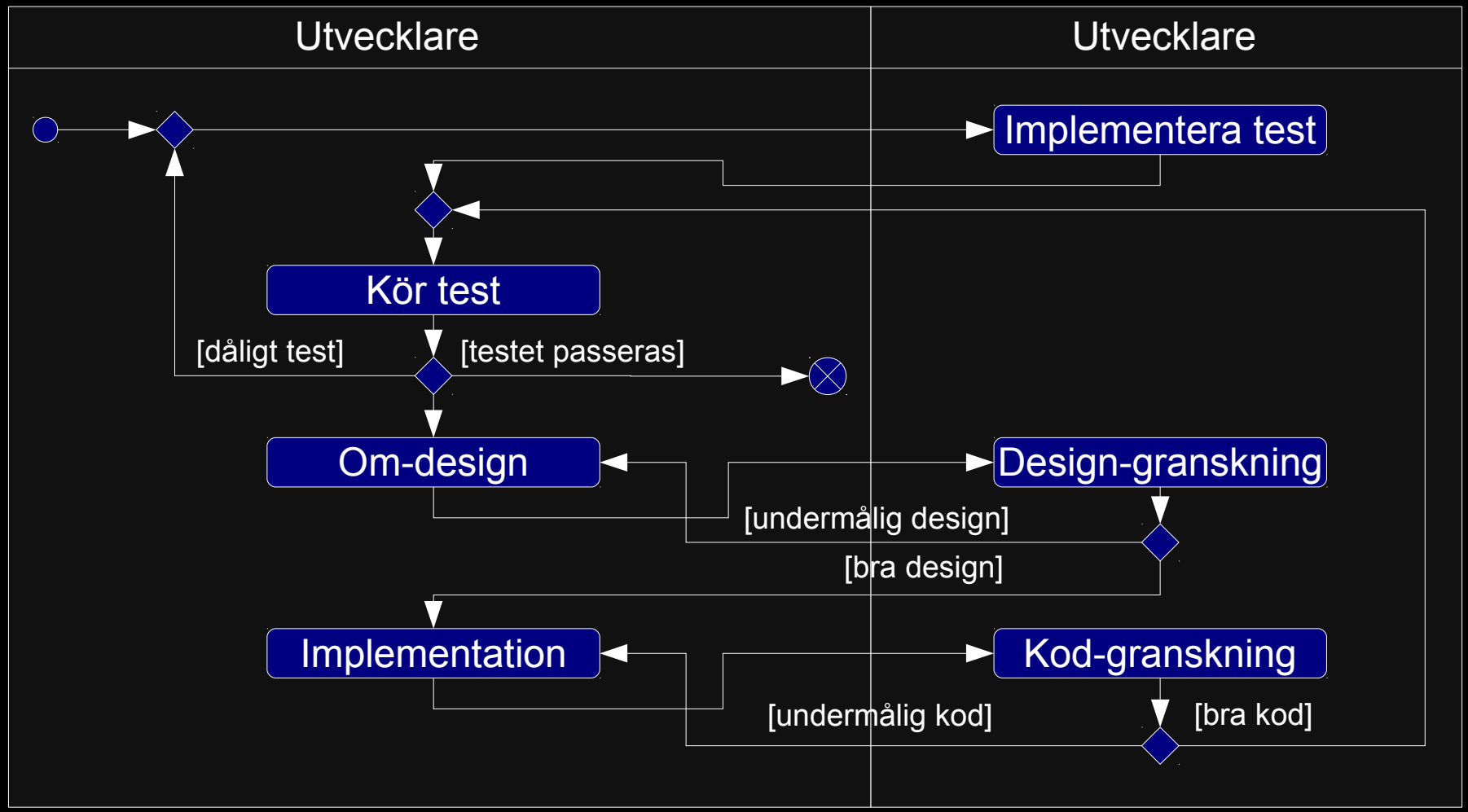
- Design- och implementationsgranskning



(UML2 Aktivitets-diagram)

Test-processer (exempel 3)

- Design- och implementationsgranskning



(UML2 Aktivitets-diagram)