Kostnad for å rette feil

Kostnad for å rette feil		Tidspunkt når feil blir oppdaget				
		Krav-spesifikasjon	Arkitektur	Konstruksjon	Systemtest	Etter release
Feil introdusert	Krav-spesifikasjon	1x	3x	5-10x	10x	10-100x
	Arkitektur		1x	10x	15x	25-100x
	Konstruksjon			1x	10x	10-25x

(NIST, M. Newman, 2002)

Hva er testing?

Helt enkelt: For en gitt input forventer vi en gitt output

Hensikten er:

- Å vise at et program gjør hva programmet er ment å gjøre
- Å oppdage feil før programmet blir tatt i bruk

Testing viser bare feil som du oppdager under kjøring av testen. Den kan ikke vise at det ikke er flere gjenstående feil

Testing er en del av en mer generell verifikasjons- og valideringsprosess.

Verifikasjon vs validering

- Verifikasjon:
  - Utvikles produktet riktig?
    - Stemmer det med kravspesifikasjonen?
    - Fokus på komponent/delsystem
- Validering:
  - Har vi utviklet riktig produkt?
    - Programvaren bør gjøre det brukerne virkelig ønsker
    - Fokus på hele systemet

Generelle retningslinjer for testing

- Velg input som tvinger systemet til å generere alle feilmeldinger
- Design input som fører til overbelastning (”overflow”)
- Gjenta samme input eller serier av input en rekke ganger
- Prøv å framprovosere ugyldig output
- Prøv å tvinge fram beregningsresultater som er for stort eller for lite

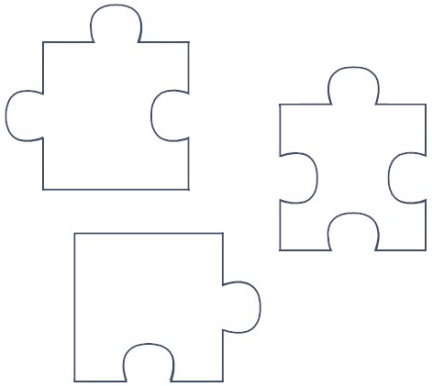
Testfaser

- Utviklingstesting. Testing foregår under utviklingsfasen
- ”Release” testing. Tester en komplett versjon av systemet før det blir satt i produksjon
- Brukertesting. Brukerne eller potensielle brukere tester systemet i egne omgivelser

Utviklingstesting

- **Enhetstesting** (unit testing). Individuelle programenheter eller objektklasser testes. Fokus på å teste funksjonaliteten til objekter og metoder.
- **Integrasjonstesting**. Ulike individuelle enheter er integrert til sammensatte komponenter. Fokus på å teste grensesnittet til komponenter. Kalles også komponenttesting.
- **Systemtesting**. Noen eller alle komponentene i et system er integrert og systemet testes som en helhet. Fokus på å teste interaksjoner mellom komponenter.

Enhetstesting



- Enhetstesting er prosessen å teste individuelle komponenter isolert
- Enheter (Units) kan være
  - Individuelle metoder i et objekt
  - Objektklasser med attributter og metoder
  - Sammensatte komponenter med definert grensesnitt

Testing av objektklasser

- Komplett testing av en klasse involverer
  - Testing av alle operasjoner (metoder) assosiert med et objekt av klassen
  - Sette verdier og teste alle attributter
  - Teste objektet i alle mulige tilstander
- Arvemekanisme gjør det vanskeligere å designe objektklassetester, siden vi på forhånd ikke vet hvor informasjonen som vi vil teste befinner seg

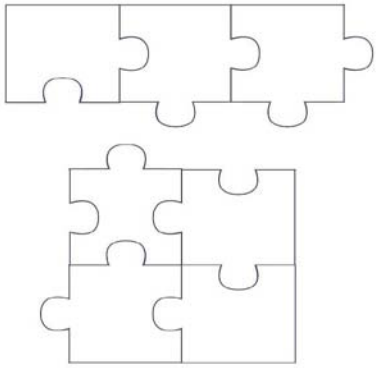
Automatisk testing

- Enhetstesting bør automatiseres så langt det lar seg gjøre, slik at tester kan kjøres uten manuell innblanding
- I automatisk enhetstesting brukes et automatiserings-rammeverk (for eksempel Junit) for å skrive og kjøre testene

To typer enhetstester

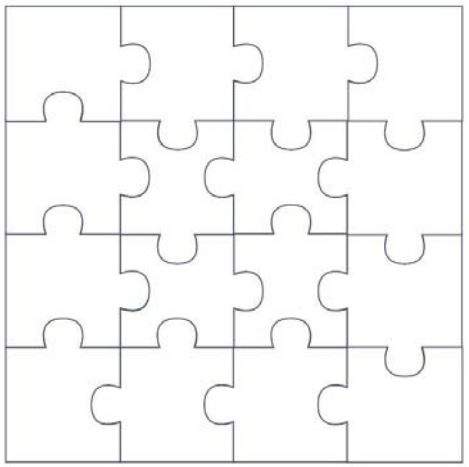
- Tester normal bruk, for å sjekke om komponenten oppfører seg som forventet
- Tester unormal bruk, for eksempel unormal input for å sjekke at komponenten ikke krasjer

Komponentbasert testing – Grensesnitt testing - Integrasjonstesting



- Programvarekomponenter er ofte sammensatt av mange objekter som samhandler
- Funksjonaliteten til objektene aksesseres gjennom deres definerte grensesnitt
- Å teste sammensatte komponenter fokuserer på at grensesnittet ”oppfører” seg i henhold til spesifikasjonen
  - Kan anta at enhetstester på alle individuelle objekter som utgjør komponenten allerede er testet

Systemtesting



- Systemtesting involverer integrerte komponenter som til sammen skaper en versjon av systemet og som tester det integrerte systemet
- Fokuset i systemtesting er å teste samhandlingen mellom komponenter (eller sammensatte komponenter)
- Systemtesting sjekker at komponentene er kompatible, at de samhandler korrekt, og at de sender riktige data til riktig tid gjennom grensesnittene.

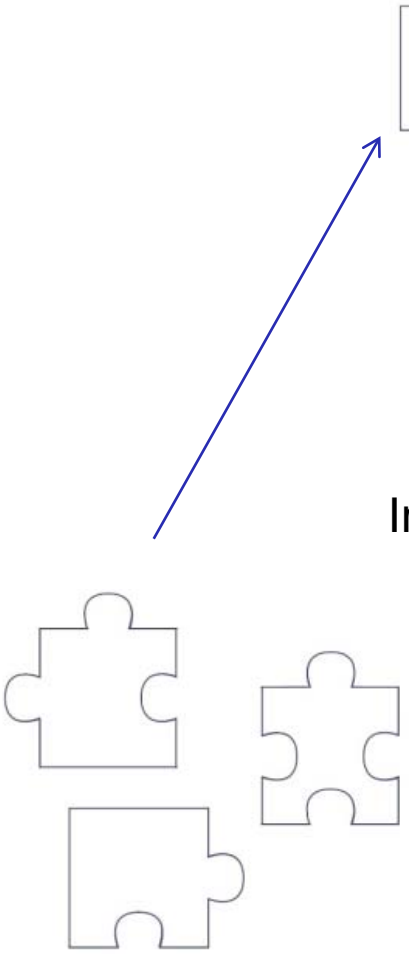
System- og komponentbasert testing

- Under systemtesting kan gjenbruk av komponenter og tidligere ferdige systemer integreres med nye komponenter.
- Komponenter som er utviklet av ulike team integreres.
  - Ofte utføres systemtesting av et separat testteam uten involvering fra utviklere

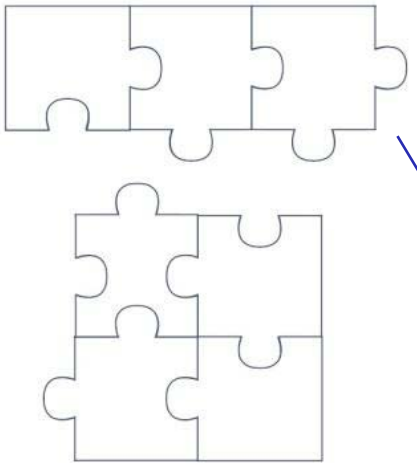
Akseptansetesting

- Evaluere systemet som leveres
- I XP bruker man ofte funksjonell testing av brukerhistorier som akseptansetester
- Fokus: Verifisere at systemet som er utviklet faktisk er det kunden vil ha
- Mål : Bekrefte at systemet imøtekommer kundens krav og er klar til bruk

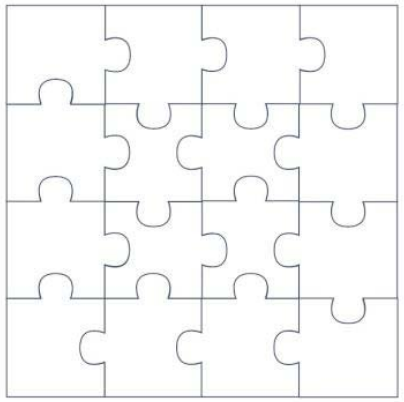
Testprosessen



Enhetstesting



Integrasjonstesting



Systemtesting

Akseptansetesting

Testdrevet utvikling

- Testdrevet utvikling (test-driven development (TDD)) er en tilnærming der testing og koding gjøres parallelt
- Tester skrives gjerne før kode, og ”passere” testene er kritisk for videre utvikling og koding
- Kode utvikles i steg (inkrementelt), sammen med en test for hvert inkrement. Går ikke til neste inkrement før koden ”passerer” testen.
- TDD ble introdusert som en del av smidig metodikken i XP. Men TDD kan også brukes i plandrevet utviklingsprosess

Prosessaktiviteter i testdrevet utvikling

- Start med å identifisere funksjonaliteten som kreves i et inkrement. Normalt lite og kan implementeres i få linjers kode
- Skriv en test for denne funksjonaliteten og lag testen som automatisk test
- Kjør testen, sammen med andre tester som er implementert. Initielt er ikke koden skrevet for dette inkrementet, slik at testen vil feile
- Implementer funksjonaliteten, og kjør testen på nytt.
- Når alle testene er OK (ikke feiler), starter implementering av funksjonalitet i neste inkrement

Fordeler ved testdrevet utvikling

- Dekning av kode
  - All segment av kode har minst en assosiert test
- Regresjonstesting
  - En regresjonstest-pakke utvikles inkrementelt og samtidig med at programmet utvikles
- Enklere å finne feil (”debugging”)
  - Når en test feiler, er det enkelt å finne ut hvor feilen ligger
- Systemdokumentasjon
  - Testene er i seg selv en form for dokumentasjon som beskriver hva koden gjør

Regresjonstesting

- Fokuserer på å finne feil etter en større kodeendring
  - Søker å finne feil som var der tidligere
  - Kjører tidligere tester om igjen
    - Fordel med automatiserte tester
  - Testene må være godkjent før kodeendringen blir godtatt (”commited”)

Black-box testing

- **Black-box testing** er en metode (eller strategi) som tester funksjonaliteten til et program i motsetning til den interne strukturen eller kildenkoden
- Kunnskap om intern struktur eller programmering er ikke nødvendig
- Tester er bygd rundt spesifikasjoner og krav, dvs. hva programmet er ment å gjøre
- Bruker ekstern beskrivelse av programvaren, med spesifikasjoner, krav og design for å lage testene
- Black-box testing kan brukes på alle nivåer av programvaretesting:
  - Enhetstesting, integrasjonstesting, systemtesting og brukertesting

Black-box testing

Fordeler med black-box testing

- Krever ikke programmeringskunnskaper
- Kan bruke uavhengige testere som ikke har vært involvert i utviklingen
- Tvinger testere til å velge et subset med tester som er effektive til å finne feil
- Unngår at testere gjør samme antagelser som programmere
- Testdataene er uavhengig av implementasjonen
- Resultater kan tolkes uten å kjenne til detaljer ved implementasjonen
- Testing gjøres fra en brukerperspektiv og kan dermed avsløre mangler i kravspesifikasjonen
- Planleggingen av testingen kan påbegynnes tidlig

Fordeler med black-box testing

- Krever ikke programmeringskunnskaper
- Kan bruke uavhengige testere som ikke har vært involvert i utviklingen
- Tvinger testere til å velge et subset med tester som er effektive til å finne feil
- Unngår at testere gjør samme antagelser som programmere
- Testdataene er uavhengig av implementasjonen
- Resultater kan tolkes uten å kjenne til detaljer ved implementasjonen
- Testing gjøres fra en brukerperspektiv og kan dermed avsløre mangler i kravspesifikasjonen
- Planleggingen av testingen kan påbegynnes tidlig

Ulemper med black-box testing

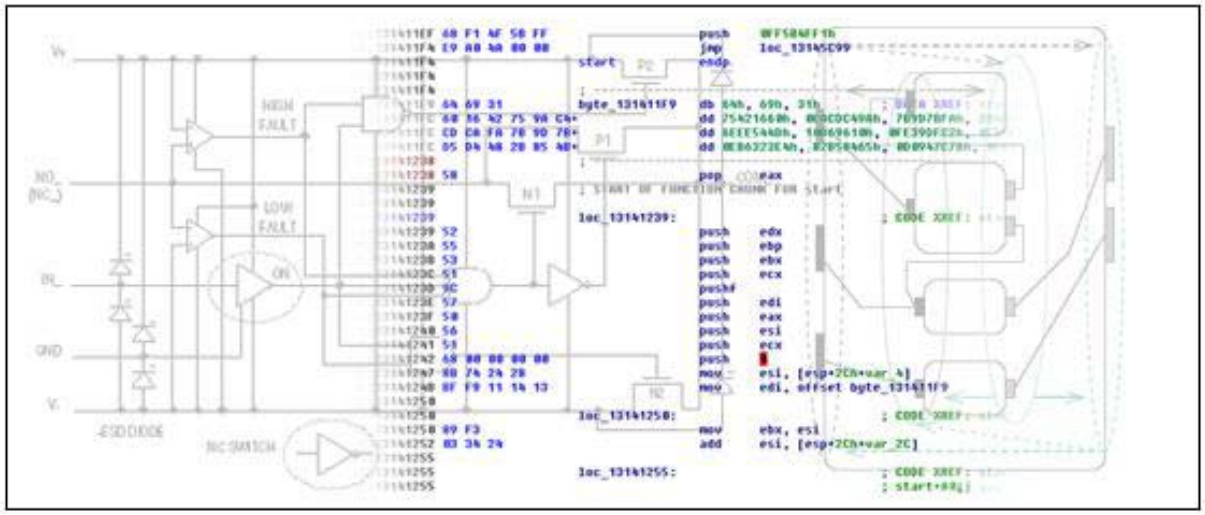
- Kan kun teste et lite antall input og mange stier og tilstander vil derfor forbli utestet
- Man er aldri sikker på hvor mye av systemet eller hvilke deler av systemet som er testet
- Kan i noen tilfeller være redundante om utviklerne allerede har kjørt
- tilsvarende tester
- Selv om man kan planlegge disse testene tidlig kan de som regel ikke anvendes før i senere testfaser

White-box testing

- **White-box testing** (glass-box testing, strukturell testing) tester interne strukturer eller kode i et program, i motsetning til dens funksjonalitet
- I white-box testing kreves og brukes programmeringsferdigheter til å designe testene
- White-box testing brukes vanligvis under enhetstesting, men kan også brukes i integrasjons- og systemtesting.
- Kan avdekke mange feil eller problemer, men oppdager ikke ting som ikke er implementert i henhold til kravspesifikasjonen, heller ikke manglende krav

White box testing

Gitt input



Forventet output

Systemet

Fordeler med white-box testing

- Kan utføres i et tidlig stadie -> trenger ikke vente på grensesnitt (f.eks. GUI)
- Testingen er mer grunding, systematisk og strukturert
- Fordi det krever innsikt i strukturen vil man enkelt finne ut av hvilken input/data
- som mest effektivt tester programmet
- Hjelper til med å optimalisere koden
- Tvinger utvikler til å begrunne kode
- Kan skrelle vekk overflødige kodelinjer

Ulemper med white-box testing

- Krever programmeringskunnskaper
- Kan kun utføres av utviklere som har vært involvert i utviklingen
- Gjøres av utviklere da det kreves både programmeringskunnskap og kjennskap til intern struktur i programmet
- Som å studere innsiden av en motor for å forstå hvorfor bilen ikke fungerer -> gir ikke alltid riktige svar
- Kan være vanskelig å fasilitere hvis implementasjonen forandres hyppig

Appendix: Andre typer tester

- Fasilitetstesting - > utfører systemet all påkrevd funksjonalitet?
- Volumtesting -> kan systemet håndtere store datavolum?
- Stresstesting -> klarer systemet å håndtere høy påkjenning?
- Utholdenhetstesting -> vil systemet klare kontinuerlig arbeid over lenger tid?
- Brukbarhetstesting -> kan brukere bruke systemet på en enkel måte?
- Sikkerhetstesting -> klarer systemet å takle angrep?
- Ytelsestesting -> hvor god er responstiden?
- Lagringstesting -> er det noen uforventede datalagringsproblemer?
- Konfigurasjonstesting -> vil systemet fungere på alle typer av påtenkt maskinvare?
- Installasjonstesting -> klarer vi å installere systemet?
- Pålitelighetstesting -> hvor pålitelig er systemet over tid?
- Gjenopprettingstesting -> hvor bra klarer systemet å gjenopprette fra feil?
- Vedlikeholdstesting -> hvor vedlikeholdbart er systemet?
- Dokumentasjonstesting -> hvor presis, dekkende, brukbar etc. er dokumentasjonen?
- Operasjonstesting -> er operatørenes instruksjoner dekkende og forståelige?
- Regresjonstesting -> gjenta alle gamle tester ved hver systemendring