Kapittel 1

False-fail result: Et testresultat der en defekt rapporteres, selv om det ikke finnes noen slik defekt i testobjektet.

Feil (feil) En menneskelig handling som gir et feil resultat.

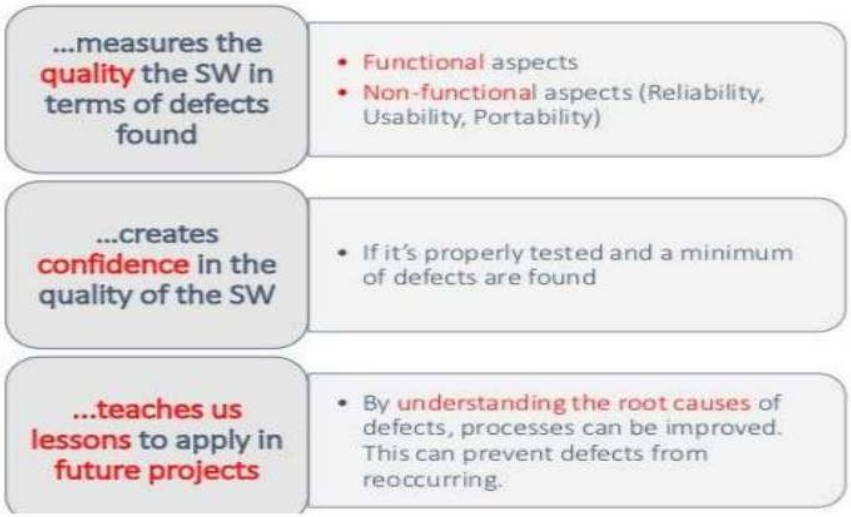
Defekt (feil, feil) En feil i en komponent eller et system som kan føre til at komponenten eller systemet ikke utfører den nødvendige funksjonen, for eksempel en feil setning eller datadefinisjon. En defekt, hvis den oppstår under utførelsen, kan føre til svikt i komponenten eller systemet.

Feil Avvik fra komponenten eller systemet fra forventet levering, service eller resultat.

Falskt bestått resultat: Et testresultat som ikke klarer å identifisere tilstedeværelsen av en defekt som faktisk er tilstede i testobjektet.

Begge årsakene til feil gir defekter (= feil, feil) i koden.

- Defekter, hvis de utføres, kan føre til feil i SW-systemet (systemet vil ikke gjøre det det skal).
- Feil kan påvirke brukerne av SW-systemet alvorlig, dvs.: • Bremsepedalen fungerer ikke i noen Biler
- Feilberegninger i finansielle SW-systemer



Testens rolle:

- For å redusere risikoen for problemer under drift
- For å sjekke om SW-systemet oppfyller:
- Juridiske krav
- Bransjespesifikke standarder

For å lære mer om SW-systemet

Testing av aktiviteter:

- **Testplanlegging:** I testplanlegging etablerer (og oppdaterer) vi omfanget, tilnærmingen, ressursene, tidsplanen og spesifikke oppgaver i de tiltenkte testaktivitetene som utgjør resten av testprosessen. Testplanlegging bør identifisere testelementer, funksjonene som skal testes og ikke testes, rollene og ansvaret til deltakerne og interessentene, forholdet mellom testerne og utviklerne av testelementene, i hvilken grad testingen er uavhengig av utviklingen av disse arbeidselementene (se avsnitt 1.5 nedenfor), testmiljøene som kreves, passende testdesignteknikker, inngangs- og utgangskriterier, og hvordan vi skal håndtere prosjektrisiko knyttet til testing. Testplanlegging produserer ofte, som et leveringsprodukt, en testplan. Testplanlegging er diskutert i kapittel 5.
- **Testkontroll:** Selv om testplanlegging er viktig, går ikke alltid ting etter planen. I testkontroll utvikler og gjennomfører vi korrigerende tiltak for å få et testprosjekt tilbake på sporet når vi avviker fra planen. Testkontroll er omtalt i kapittel 5.
- **Testanalyse:** I testanalyse identifiserer vi hva vi skal teste, og velger testforholdene vi trenger lokk. Disse betingelsene er ethvert element eller hendelse som vi kan og bør verifisere ved hjelp av en eller flere

testtilfeller. Testbetingelser kan være funksjoner, transaksjoner, funksjoner, kvalitetsattributter, kvalitetsrisikoer eller strukturelle elementer. Testanalyse er omtalt i kapittel 4.

- **Testdesign:** I testdesign bestemmer vi hvordan vi skal teste det vi bestemte oss for å teste under testen analyse. Ved hjelp av testdesignteknikker transformerer vi disse generelle testbetingelsene og de generelle testmålene fra testplanen til konkrete testtilfeller på passende detaljnivå. Testdesign generelt, og spesifikke testdesignteknikker, er diskutert i kapittel 4.
- **Testimplementering:** Ved testimplementering utfører vi de resterende aktivitetene som kreves for å være klare for testutførelse, som å utvikle og prioritere testprosedyrene våre, lage testdata og sette opp testmiljøer. Disse elementene i testgjennomføringen er dekket i kapittel 4. I mange tilfeller ønsker vi å bruke automatisert testutførelse som en del av vår testprosess. I slike tilfeller inkluderer testimplementering å klargjøre testseler og skrive automatiserte testskript. Disse elementene i testimplementering er dekket i kapittel 6.
- **Testutførelse:** I testkjøring kjører vi testene våre mot testobjektet (også kalt systemet som testes).
- **Kontrollere resultater:** Som deltestutførelse ser vi de faktiske resultatene av testtilfellet, konsekvenser og utfall. Disse inkluderer utdata til skjermer, endringer i data, rapporter og kommunikasjonsmeldinger som sendes ut. Vi må sammenligne disse faktiske resultatene med forventede resultater for å bestemme bestått/ikke bestått-status for testen. Definisjon av forventede resultater er diskutert i kapittel 4, mens styring av testutførelse, inkludert kontroll av resultater, er diskutert i kapittel 5.
- **Evaluering av exit-kriterier:** På et høyt nivå er exit-kriterier et sett med betingelser som gjør at en del av en prosess kan fullføres. Avslutningskriterier defineres vanligvis under testplanlegging ved å samarbeide med prosjekt- og produktinteressenter for å balansere kvalitetsbehov mot andre prioriteringer og ulike prosjektbegrensninger. Slike kriterier sikrer at alt som skal gjøres er gjort før vi erklærer et sett med aktiviteter ferdige. Nærmere bestemt hjelper testavslutningskriterier oss med å rapportere resultatene våre (ettersom vi kan rapportere fremgang mot kriteriene) samt hjelpe oss med å planlegge når vi skal slutte å teste. Etablering og evaluering av exit-kriterier er omtalt i kapittel 5. Rapportering av testresultater: I rapportering av testresultater ønsker vi å rapportere fremgangen vår mot utgangskriterier, som beskrevet ovenfor. Dette innebærer ofte detaljer knyttet til status for testprosjektet, testprosessen og kvaliteten på systemet som testes. Vi vil diskutere rapportering av testresultater i kapittel 5.
- **Testlukking:** Testavslutning innebærer å samle inn testprosessdata relatert til de ulike fullførte testaktivitetene for å konsolidere vår erfaring, gjenbrukbar testvare, viktige fakta og relevante beregninger. Testavslutning er omtalt i avsnitt 1.4 nedenfor.

fossefall :need wish , policy law , user req , system req , global design , detailed design , implementation , testing

exit kriterier : er sett med betingelser som lar prosessen utføre

test plan og kontroll , analyse design , implementasjon , utførelse , ferdigstille

Testdesignspesifikasjon Et dokument som spesifiserer testbetingelsene (dekningselementene) for et testelement, den detaljerte testtilnærmingen og identifiserer de tilknyttede testtilfellene på høyt nivå.

Testkontroll En teststyringsoppgave som omhandler å utvikle og bruke et sett med korrigerende handlinger for å få et testprosjekt på rett spor når overvåkingen viser et avvik fra det som var planlagt.

Testtilfelle Et sett med inngangsverdier, utførelsesforutsetninger, forventet resultater og utførelse etter betingelser, utviklet for et bestemt mål eller testforhold, for eksempel for å utøve en bestemt programbane eller for å verifisere samsvar med et spesifikt krav.

Testmål En grunn eller et formål for å utforme og utføre en test.

Testing Prosessen som består av alle livssyklusaktiviteter, både statiske og dynamiske, som dreier seg om planlegging, klargjøring og evaluering av programvareprodukter og relaterte arbeidsprodukter for å fastslå at de oppfyller spesifiserte krav, for å demonstrere at de er egnet til formålet og for å oppdage defekter.

Krav En tilstand eller evne som en bruker trenger for å løse et problem eller oppnå et mål som må oppfylles eller besittes av en system- eller systemkomponent for å tilfredsstille en kontrakt, standard, spesifikasjon eller annen formelt pålagt dokument.

Gjennomgå En evaluering av et produkt- eller prosjektstatus for å fastslå

avvik fra planlagte resultater og å anbefale forbedringer. Eksempler inkluderer ledelsesgjennomgang, uformell gjennomgang, teknisk gjennomgang, inspeksjon og gjennomgang.

Feilsøking Prosessen med å finne, analysere og fjerne årsakene til feil i programvaren.

Bekreftelsestesting (re-testing) Testing som kjører testtilfeller som mislyktes forrige gang de ble kjørt, for å bekrefte at korrigerende handlinger er vellykkede. [Merk: Mens ordlisten bruker re-testing som det foretrukne begrepet, er denne preferansen i utakt med vanlig bruk og med den faktiske engelskspråklige definisjonen av 're-testing'.]

Syv testprinsipper

p1 : tilstedeværelse av defekter beviser ikke at det ikke finnes defekter(feil)

Principle 1:	**Testing shows presence of defects**	Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness.
Principle 2:	**Exhaustive testing is impossible**	Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, risk analysis and priorities should be used to focus testing efforts.
Principle 3:	**Early testing**	To find defects early, testing activities shall be started as early as possible in the software or system development life cycle, and shall be focused on defined objectives.
Principle 4:	**Defect clustering**	Testing effort shall be focused proportionally to the expected and later observed defect density of modules. A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures.
Principle 5:	**Pesticide paradox**	If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new defects. To overcome this 'pesticide paradox', test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to find potentially more defects.
Principle 6:	**Testing is context dependent**	Testing is done differently in different contexts. For example, safety-critical software is tested differently from an e-commerce site.
Principle 7:	**Absence-of-errors fallacy**	Finding and fixing defects does not help if the system built is unusable and does not fulfil the users' needs and expectations.

P2: fullstensig testing er mulig kun i trivielle caser , insteden bør risikoanalyse og prioritering bli fokus for testinsatsen

P3 : tidlig testing bør skje tidlig: testaktivitetene bør skje tidlig som mulig

p4 : moduler inyneholder flest feil og kan medføre driftsfeil

p5: retesting finner ikke nye feil , insteden bør revisjon og gjennomgang brukes , skriv ulike tester for å teste ulike systemkomponenter

p6 :testing er ulik i forskjellige kontekster

p7 : å finne feil eller defekter er ikke vist dersom systemet er ubrukelig og ikke oppfyller brukerkarv og behov

Grunnleggende testprosess

- Planlegging og kontroll
- Analyse og design
- Implementering og gjennomføring
- Evaluering av exit-kriterier og rapportering
- Aktiviteter for stenging av tester

planlegging og kontroll ¨
analyse og design
test impleementasjon
utførelse
ferdigstille

dette er i følge pensum

exit kriterier er et sett med betingelser som gjør at prossesen kan utføres

plan kontroll , ANALYSE DESIGN , Implementasjon , utførelse , ferdigstille

En god tester trenger:

- Nysgjerrighet
- Profesjonell pessimisme
- Oppmerksomhet på detaljer
- Gode kommunikasjonsevner
- Erfaring med gjetting
- Å kommunisere feil og feil på en konstruktiv måte: faktafokuserte rapporter og gjennomgang av funn

en god tester: nysgjerrig , gjetting , finne detaljer , proff pessimisme , kommunikasjonsevner (samarbeide)

Teststrategi En overordnet beskrivelse av testnivåene som skal utføres og testingen innenfor disse nivåene for en organisasjon eller et program (ett eller flere prosjekter).

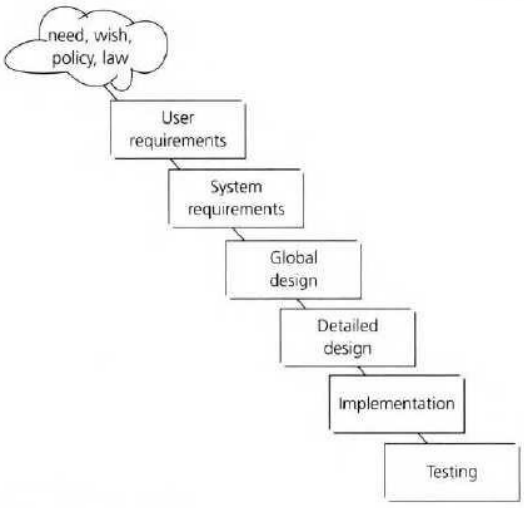
Uttømmende testing (fullstendig testing) En testtilnærming der testpakken omfatter alle kombinasjoner av inngangsverdier og forutsetninger.

Testkjøring Prosessen med å kjøre en test på komponenten eller systemet som testes, og produsere faktiske resultater.
Testtilnærming Implementering av teststrategien for et spesifikt prosjekt. Det inkluderer vanligvis beslutninger som tas som følger basert på (test)prosjektets mål og risikovurderingen som er utført, utgangspunkt for testprosessen, testdesignteknikkene som skal brukes, avslutningskriterier og testtyper som skal utføres.

Kapittel 2

Verifikasjon Bekreftelse ved undersøkelse og gjennom fremleggelse av objektive bevis på at spesifiserte krav er oppfylt.
Validering Bekreftelse ved undersøkelse og gjennom fremleggelse av objektive bevis på at krav til en bestemt tiltenkt bruk eller anvendelse er oppfylt.

Fossefall-modell

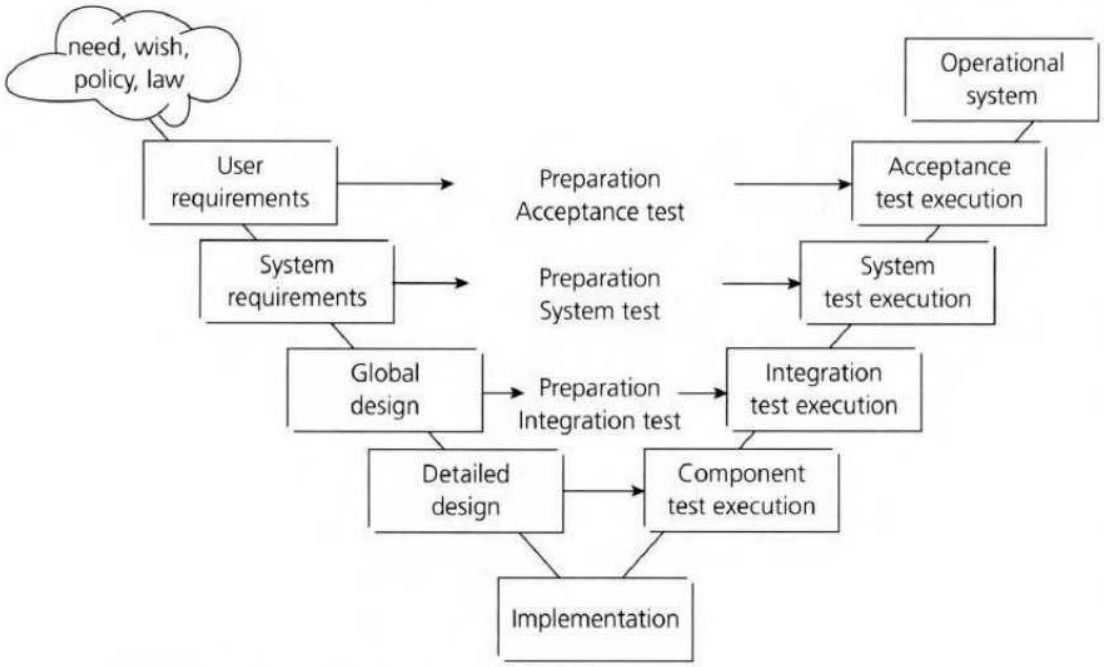


Fossefal modellen
need , wish , policy law , user req , system req , global design , detailed design , implmentation , testing

V-modell Et rammeverk for å beskrive programvareutviklingens livssyklusaktiviteter fra kravspesifikasjon til vedlikehold. V-modellen illustrerer hvordan testaktiviteter kan integreres i hver fase av programvareutviklingens livssyklus.

v modellen har verifisering tester , v modellen sørger for å integrere testing til hver utviklingsaktivitet i utviklingsprosessen

testing skal integreres i de ulike stege / fasene i en utviklingsprosessen det bør skje tidlig
hver testaktivitet har en korresponderende mål som er spesifikk for det nivået



• Testing må begynne så tidlig som mulig i livssyklusen.
• Testing kan integreres i hver fase av livssyklusen.
• Innenfor V-modellen foregår valideringstesting
spesielt under:

- de tidlige stadiene, det vil si gjennomgang av brukerkravene
- og sent i livssyklusen, det vil si under testing av brukeraksept

testing bør skje tidlig
integreres i hver fase
og det bør skje en verifisering test i v modellen (inkludert)

verifiserer funksjonen til programvare komponenten = komponent testing

De fire testnivåene som brukes, hver med sine egne mål, er:

• ***komponenttesting***: søker etter defekter i og verifiserer funksjonen til programvarekomponenter (f.eks. moduler, programmer, objekter, klasser osv.) som kan testes separat;

Komponenttesting inkluderer testing av funksjonalitet og spesifikke ikke-funksjonelle egenskaper, for eksempel:

- ressursatferd (f.eks. minnelekkasjer)
- Testing av robusthet
- strukturell testing (f.eks. grendekning).

komponent tester detekter og verifiserer funk til komponent\ene

Alle materialer som er aktuelle for komponenten som testes:

• Spesifikasjon av komponenten • Programvaredesign
• Datamodellen
• I tillegg til selve koden

Skrive kode for å teste prosjektkoden

• Stubber, drivere og simulatorer kan brukes.

Stubber:

Kode som erstatter en kalt komponent for å simulere formålet (dvs. "hardkodede" data for å erstatte data fra en database).

Drivere:

Kode som erstatter en annen programvarekomponent for å kalle komponenten som testes.

Komponenttesting involverer vanligvis programmereren som skrev koden.
Feil fikses så snart de blir funnet, uten formell registrering av hendelser. TDD
testdrevet utvikling
• Forbered og automatiser testtilfeller før koding
• brukt i XP (ekstrem programmering)

• ***integrasjonstesting***: tester grensesnitt mellom komponenter, interaksjoner til forskjellige deler av et system som et operativsystem, filsystem og maskinvare eller grensesnitt mellom systemer;

Test interaksjoner med ulike deler av et system, for eksempel:
• Operativsystemet
• filsystem
• maskinvare
• grensesnitt mellom systemer

testnivå : unit , integration , akseptanse system

unit : tester system funksjonaliteten

Testgrunnlag:
• Programvare- og systemdesign
  • Systemarkitekturen
• Arbeidsflyter/brukstilfeller

Elementet som testes inkluderer
• bygg som inkluderer noen eller alle komponentene i systemet
• Databasens elementer
• Systeminfrastruktur
• Grensesnitt mellom komponenter eller objekter
• Systemkonfigurasjon
• Konfigurasjonsdata

iintegrasjonstest : tester grensesnitt mellom komponentene og interaksjoner til de ulike delene av systemet.

akseptanse

non funk : tester egenskaper ved en programvare

funksjonnel tester eksterne / interne oppførselen til systemet

strukturell : tester strukturen og arkitekturen til systemet

Typer integrasjonstesting
Komponent integrasjon
Tester interaksjonene mellom programvarekomponenter gjøres etter komponenttesting
Systemintegrasjon
Tester interaksjonene mellom ulike systemer gjøres etter systemtesting.

Tilnærminger og ansvar
• Start integrasjonen med de komponentene som forventes å forårsake flest problemer.
• For å redusere risikoen for sen feiloppdagelse, bør integrasjon normalt være inkrementell i stedet for "big bang".
• Både funksjonelle og strukturelle tilnærminger kan brukes.
• Ideelt sett bør testere forstå arkitekturen og påvirke integrasjonsplanleggingen.
• Kan gjøres av utviklerne eller av et eget team.

• ***Systemtesting***: opptatt av oppførselen til hele systemet/produktet som definert av omfanget av et utviklingsprosjekt eller produkt. Hovedfokuset for systemtesting er verifisering mot spesifiserte krav;

Testing av oppførselen til hele systemet som definert av prosjektets omfang.

Testgrunnlag:
• Systemkravspesifikasjon, både funksjonell og ikke-funksjonell
• Forretningsprosesser

systemtesting er opptatt av oppførselen til sysetemet

vedlikeholdsetest : tester drift

• Risikoanalyse
• Bruksområder
• Andre overordnede beskrivelser av systemets oppførsel, interaksjoner med OS/systemressurser
• Krav kan eksistere som tekst og/eller modeller.
• Testere må også håndtere ufullstendige eller udokumenterte krav.

kompetanse verifiserer funk til komponentene og ser etter defekter

Test objekter:
• Hele det integrerte systemet
• Bruksanvisninger
• Bruksanvisninger
• Informasjon om systemkonfigurasjon
• Konfigurasjonsdata

Testmiljøet skal samsvare med produksjonsmiljøet så mye som mulig.
• For det første, den mest egnede black-box-teknikken
• Deretter white-box-teknikk for å vurdere grundigheten av testingen Et uavhengig testteam kan være ansvarlig for testingen Graden av uavhengighet er basert på gjeldende risikonivå

akseptanse er valideringstest med hensyn til bruker behov og krav avgjør om systemet kan godtas eller ikke

• ***Aksepttesting***: Valideringstesting med hensyn til brukerbehov, krav og forretningsprosesser utført for å avgjøre om systemet skal godtas eller ikke.

Spørsmålene som skal besvares:
• Kan systemet frigjøres?
• Hva er de utestående risikoene?
• Har utviklingen oppfylt sine forpliktelser?

funk : tester det ekster (blackbox)oppførselen til systemet
nonfunk tester egenskaper av programvare
strukturell : tester struktur og arkitekturen til systemet

Målet er å skape tillit til
• Systemet
• Ikke-funksjonelle egenskaper ved systemet

Testgrunnlag:
• Spesifikasjon av brukerkrav
• Bruksområder
• Spesifikasjon av systemkrav
• Forretningsprosesser
• Risikoanalyse

akseptanse : verifiserings test avgjør om systemet kan godtas eller ikke

aksept : verifiseringstest med hensyn til bruker behov krav og avgjør om systemet skal godtas eller ikke

Typer:
Brukeraksepttesting – valider egnetheten for bruk av systemet av brukere.
• Operasjonell testing, vanligvis utført av systemadministratorene:
  • testing av sikkerhetskopiering/gjenoppretting
  • Katastrofegjenoppretting
  • Brukeradministrasjon
  • Vedlikeholdsoppgaver
  • Periodiske kontroller av sikkerhetssårbarheter
• Kontrakts- og forskriftsaksepttesting utført mot en kontrakts akseptkriterier (dvs. statlige, juridiske eller sikkerhetsforskrifter)
• Alfa- og betatesting

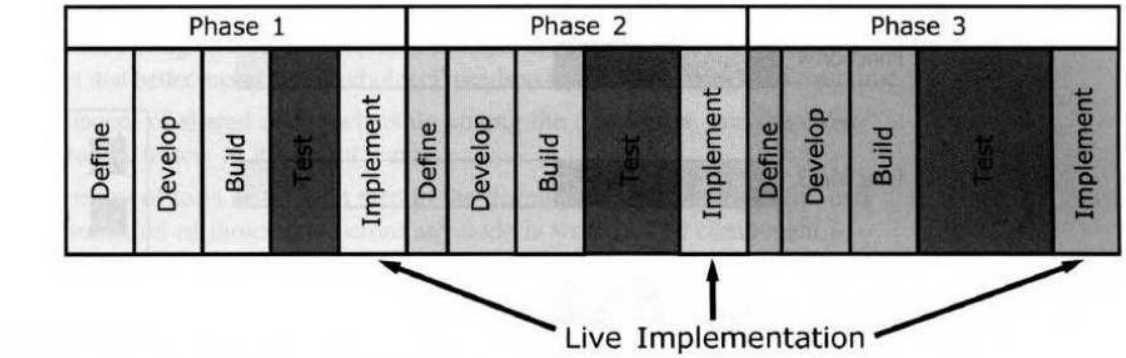
• Alfatesting utføres på utviklingsorganisasjonens nettsted.
• Betatesting (felttesting), utføres av folk på sine egne lokasjoner.

Begge utføres av potensielle kunder, ikke utviklerne av produktet.

inkrementell : syklusen hvor prosjektet er delt inn i en rekke trinn hvor hver trinn utgjør en del av fuksjonaliteten

iterativ : syklus hvor prosjektet er delt inn i store antall gjentakelser

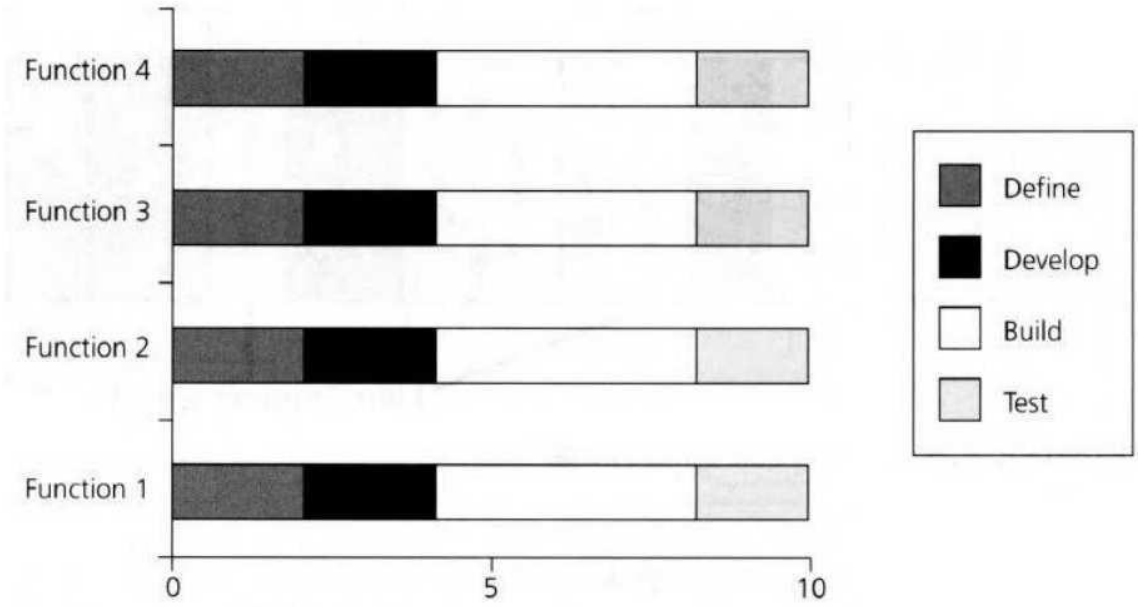
Iterativ utviklingsmodell



Inkrementell utviklingsmodell En utviklingslivssyklus der et prosjekt er delt inn i en rekke trinn, som hver leverer en del av funksjonaliteten i de generelle prosjektkravene. Kravene prioriteres og leveres i prioritert rekkefølge i riktig trinn. I noen (men ikke alle) versjoner av denne livssyklusmodellen følger hvert delprosjekt en «mini V-modell» med egne design-, kodings- og testfaser.

Iterativ utviklingsmodell En utviklingslivssyklus der et prosjekt er delt inn i et vanligvis stort antall gjentakelser. En iterasjon er en komplett utviklingssløyfe som resulterer i en utgivelse (intern eller ekstern) av et kjørbart produkt, en delmengde av sluttproduktet under utvikling, som vokser fra iterasjon til iterasjon for å bli sluttproduktet.

Rask applikasjonsutvikling



Rapid Application Development (RAD) er formelt sett en parallell utvikling av funksjoner og påfølgende integrasjon. En tidlig forretningsfokusert løsning på markedet gir tidlig avkastning på investeringen (ROD og kan gi verdifull markedsføringsinformasjon for virksomheten. Validering med utviklingsprosessen for rask utvikling er dermed en tidlig og stor aktivitet.

Smidig utvikling

Smidig programvareutvikling er en gruppe programvareutviklingsmetoder basert på iterativ inkrementell utvikling, der krav og løsninger utvikler seg gjennom samarbeid mellom selvorganiserende tverrfunksjonelle team.

Smidig utvikling gir både fordeler og utfordringer for testere. Noen av fordelene er:

- fokuset på fungerende programvare og kode av god kvalitet;
- inkludering av testing som en del av og utgangspunktet for programvareutvikling (testdrevet utvikling);
- tilgjengelighet av forretningsinteressenter for å hjelpe testere med å løse spørsmål om forventet oppførsel av systemet;
- selvorganiserende team der hele teamet er ansvarlig for kvalitet og gir testere mer autonomi i arbeidet sitt; og
- enkelhet i design som skal være lettere å teste.

Det er også noen betydelige utfordringer for testere når de går over til en smidig utviklingstilnærming.

- Testere som er vant til å jobbe med veldokumenterte krav, vil designe tester fra en annen type testgrunnlag: mindre formell og kan endres. Manifestet sier ikke at dokumentasjon ikke lenger er nødvendig eller at det ikke har noen verdi, men det tolkes ofte slik.
- Fordi utviklere gjør mer komponenttesting, kan det være en oppfatning av at testere ikke er nødvendig. Men komponenttesting og bekreftelsesbasert aksepttesting av bare forretningsrepresentanter kan gå glipp av store problemer. Systemtesting, med sitt bredere perspektiv og vekt på ikke-funksjonell testing samt ende-til-ende funksjonstesting, er nødvendig, selv om det ikke passer komfortabelt inn i en sprint.
- Testerens rolle er annerledes: siden det er mindre dokumentasjon og mer personlig Interaksjon i et smidig team må testere tilpasse seg denne arbeidsstilen, og dette kan være vanskelig for noen testere. Testere kan fungere mer som trenere i testing for både interessenter og utviklere, som kanskje ikke har mye testkunnskap.
- Selv om det er mindre å teste i en iterasjon enn et helt system, er det også et konstant tidspress og mindre tid til å tenke på testingen for de nye funksjonene.
- Fordi hvert trinn legger til et eksisterende fungerende system, blir regresjonstesting ekstremt viktig, og automatisering blir mer fordelaktig. Men bare å ta eksisterende automatiserte komponent- eller komponentintegrasjonstester kan ikke gjøre en tilstrekkelig regresjonspakke.

utforming og analyse bør skje under testaktivitetene eller testprossesen

Oppsummert, uansett hvilken livssyklusmodell som brukes, er det flere kjennetegn ved god testing:

- for hver utviklingsaktivitet er det en tilsvarende testaktivitet;
- Hvert testnivå har testmål som er spesifikke for det nivået;
- analysen og utformingen av tester for et gitt testnivå bør begynne under den tilsvarende utviklingsaktiviteten;
- Testere bør være involvert i å gjennomgå dokumenter så snart utkast er tilgjengelig i utviklingssyklusen.

tester bør være involvert dokumentasjonsgjennomgang så snart utkastet er tilgjengelig
utviklingsaktiviteter har en testaktivitet

hver testnivå har et testmål som er spesifikk for det nivået

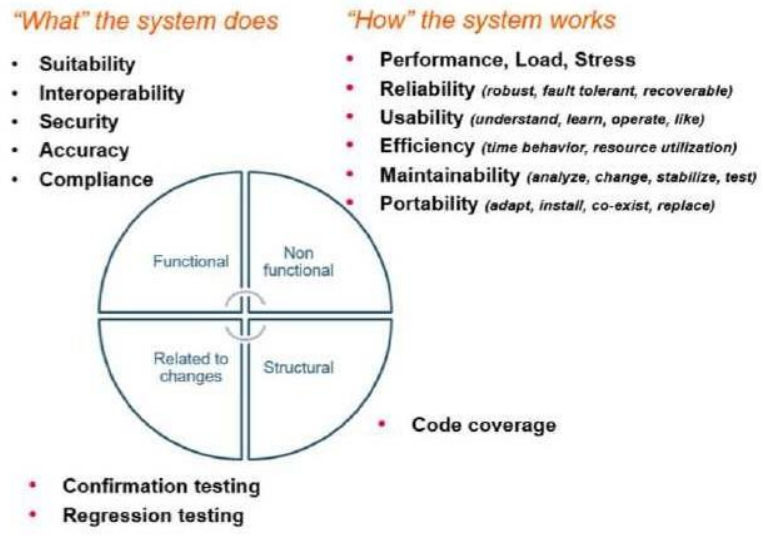
bør være involvert i gransking og dokumentasjon

analysen og utfor av testnivå bør begynne under utviklingsaktiviteten

Testtyper

utviklingsaktiviteter har vær sin test aktivitet
tester bør være involvert i dokument gjennomgang
analyse og utforming av tester for et gitt nivå bør starte under utviklingsaktiviteten

nysgjerrig
oppdage detaljer
pessimisme
flink til å finne defekter
kommunikasjonsevner



Funksjonell testing (Black box-testing)

Mål

Test hva et system skal gjøre og vurder den eksterne oppførselen til programvaren.

Testnivåer

Kan utføres på alle testnivåer

Testgrunnlag

Beskrivelsen av forventet virkemåte finnes i arbeidsprodukter som:

- Kravspesifikasjon - forretningsprosesser
- Bruksområder
- Funksjonelle spesifikasjoner
- kan være udokumentert

tester egenskaper ved en programvare

Ikke-funksjonell testing

Mål

Måling av egenskaper ved programvare som kan kvantifiseres i varierende skala: f.eks.

Testnivåer

Kan utføres på alle testnivåer

Du finner mer om dem i 'Software Engineering – Software Product Quality' (ISO 9126).

teste egenskaper av prog som kan kvantifiseres i ulik skala

Strukturell testing (white box-testing)

Mål

Måling av grundigheten av testing gjennom vurdering av dekningen av et sett med strukturelle elementer eller dekningselementer.

Testnivåer

Strukturel testing, også kjent som white-box testing, er en type programvaretesting som fokuserer på den interne strukturen og arkitekturen til et system

Kan utføres på alle testnivåer, men spesielt i komponenttesting og komponentintegrasjonstesting.
Testgrunnlag
Strukturell testing er basert på strukturen til koden så vel som arkitekturen til systemet (f.eks. et anropshierarki, en forretningsmodell eller en menystruktur)
Nærme seg
Strukturelle teknikker brukes best etter spesifikasjonsbaserte teknikker, for å hjelpe til med å måle grundigheten av testingen.
Redskapene
Dekningsmålingsverktøy vurderer prosentandelen av kjørbare elementer (f.eks. uttalelser eller beslutningsresultater) som har blitt utøvd.

Testing knyttet til endringer, bekreftelse og regresjonstesting

Bekreftelse testing konfirmasjonstest bekrefter at feilen eller defektene er dekket
Etter at en defekt er oppdaget og rettet, bør programvaren testes på nytt for å bekrefte at den opprinnelige defekten er fjernet. bekreftelses /konfirmasjon

Regresjonstesting

Gjentatt testing av et allerede testet program, etter modifikasjon, for å oppdage eventuelle defekter som er introdusert eller avdekket som følge av endringen(e).

Målsetting
For å verifisere at endringer i programvaren eller miljøet ikke har forårsaket utilsiktede bivirkninger og at systemet fortsatt oppfyller kravene.
Testnivåer
Kan utføres på alle testnivåer, gjelder funksjonell, ikke-funksjonell og strukturell testing.

Nærme seg
Omfanget av regresjonstesting er basert på risikoen for å finne feil i programvare som fungerte tidligere.
Regresjonstestsuiter kjøres mange ganger og utvikler seg generelt sakte, så regresjonstesting er en sterk kandidat for automatisering.
Hvis regresjonstestpakken er veldig stor, kan det være mer hensiktsmessig å velge et delsett for utførelse.

Vedlikehold testing
Mål
Vedlikeholdstesting utføres på et eksisterende driftssystem, og utløses av modifikasjoner, migrering eller tilbaketrekking av programvaren eller systemet.
Typer
modifikasjoner
• Planlagte forbedringsendringer (f.eks. utgivelsesbaserte)
• Korrigerende og akutte endringer (patcher)
• Endringer i miljø (operativsystem eller databaseoppgraderinger)
Migrering (f.eks. fra en plattform til en annen)
• Operasjonelle tester av det nye miljøet
• tester på den endrede
programvaren. Pensjonering
av et system
Testing av datamigrering eller arkivering hvis det kreves lange dataoppbevaringsperioder.

Omfang
Omfanget av vedlikeholdstesting er relatert til:

• Risikoen for endringen
• størrelsen på det eksisterende systemet
• størrelsen på
endringen Testnivåer
Avhengig av endringene kan vedlikeholdstesting utføres på et eller alle testnivåer og for en eller alle tester Typer. Nærme
seg
Å bestemme hvordan det eksisterende systemet kan bli påvirket av endringer kalles konsekvensanalyse, og brukes til å bestemme hvor mye regresjonstesting som skal gjøres.
Notat
Vedlikeholdstesting kan være vanskelig hvis spesifikasjonene er utdaterte eller mangler.

Kapittel 3

Dynamisk testing - krever kjøring av programvare.

Statisk testing - manuell undersøkelse og automatisert analyse av koden eller dokumentasjonen uten kjøring av programvaren som testes.

Anmeldelser - en måte å teste programvareprodukter (inkludert kode) og kan utføres i god tid før dynamisk testutførelse.

• Gjennomganger, statisk analyse og dynamisk testing har samme mål: å identifisere defekter.

• De er komplementære.

• Sammenlignet med dynamisk testing finner statiske teknikker årsaker til feil (defekter) i stedet for selve feilene.

statisk finner årsak til feil , mens dynamisk finner finner selve feilen ,

akseptanse testing : testing som valideres med hensyn til brukerkrav eller behov

Grunn til å gjøre anmeldelser

- Defekter som oppdages under gjennomganger tidlig i livssyklusen er ofte billigere å fjerne enn de som oppdages under kjøring av tester.

review /andmeldelser tidlig vil oppdage feil før og bedre enn dynamisk og det er oftere billigere å fjerne

- Anmeldelser kan finne feil og mangler, for eksempel i krav, som neppe vil bli funnet i dynamisk testing.

Verktøy (manuell + verktøystøtte)

Den viktigste manuelle aktiviteten er å undersøke et arbeidsprodukt og komme med kommentarer om det.

Gjenstand for anmeldelser

Ethvert programvarearbeidsprodukt kan vurderes, f.eks.

• Design spesifikasjoner

• kode

• Testplaner, testspesifikasjoner, testtilfeller, testskript • Brukerveiledninger

• Websider

Fordeler

• tidlig feildeteksjon og korreksjon

• Forbedringer av utviklingsproduktivitet • Reduserte utviklingstidsrammer

• Reduserte testkostnader og -tid

redusere tid og kostnader , forbedrer utviklingen produktivitet , færre feil og forbedret kommunikasjon

• Reduksjon av livstidskostnader

en god tester : pessimisme , nysgjerrighet , flink til å finne defekter , er god med detaljer

• Færre feil og forbedret kommunikasjon

Typiske feil

• Avvik fra standarder

• krav mangler

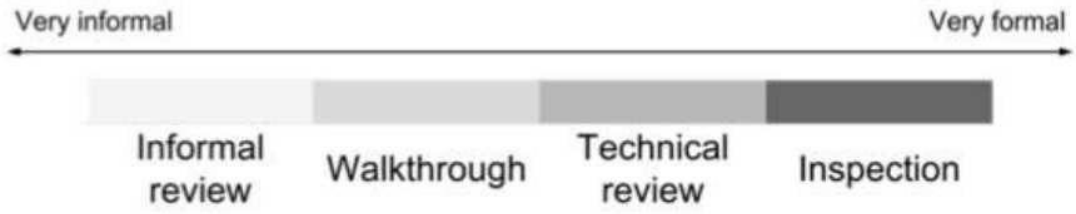
• Design feil

• utilstrekkelig vedlikehold

• feil grensesnittspesifikasjoner

Disse feilene er lettere å finne i anmeldelser enn i dynamisk testing

Ulike typer anmeldelser



Ulike typer anmeldelser varierer fra:

• veldig uformell(f.eks. ingen skriftlige instruksjoner for anmeldere)

• til veldig formell (dvs. godt strukturert og regulert)

Formaliteten til en gjennomgangsprosess er relatert til faktorer som:

• Risiko

• Prosjektets størrelse

• modenheten til utviklingsprosessen

• eventuelle juridiske eller forskriftsmessige krav

krav , prisjektstørrelse , modenhet , risiko

• Behovet for et revisjonsspor

Måten en gjennomgang gjennomføres på avhenger av det avtalte målet med gjennomgangen:

• Finn feil og mangler
• Få forståelse
• Diskusjon og beslutning ved konsensus

Faser i en formell gjennomgang faser i en formell gjennomgang

1. Planlegging
   a. Velg personell
   b. Tildele roller
   c. Definer inngangs- og utgangskriterier for mer formelle gjennomgangstyper (f.eks. inspeksjon)
   d. Velg hvilke deler av dokumentene du vil se på

2. Avspark plan , kickoff , individuell review , møtet , fiks og rapporter
   a. Distribuere dokumenter
   b. Forklare målene med gjennomgangen og gjennomgangsprosessen
   c. Forklare dokumentene til deltakerne
   d. Sjekke og diskutere inn-/utreisekriterier

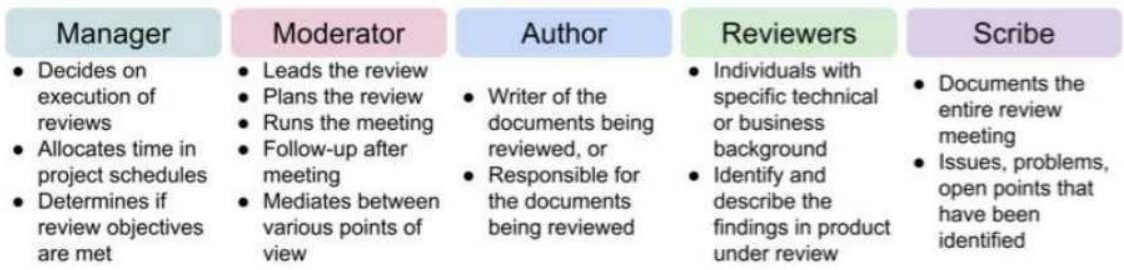
3. Individuell forberedelse
   Arbeid utført av hver av deltakerne på egen hånd før gjennomgangsmøtet, og noterer potensielle mangler, spørsmål og kommentarer.
   Hver deltaker foreslår alvorlighetsgraden av feilene. Alvorlighetsklasser: kritiske, store eller mindre

4. Gjennomgå møte
   • Diskusjon og loggføring, med dokumenterte resultater eller referater
   • Møtedeltakerne kan ganske enkelt notere mangler, komme med anbefalinger for håndtering av manglene eller ta avgjørelser om feilene. Beslutninger basert på utmeldingskriteriene
   • Undersøkelse, evaluering og opptak

5. Bearbeide
   • Å fikse feil funnet, vanligvis utført av forfatteren.

6. Oppfølging
   • Sjekk at feil er utbedret
   • samle inn beregninger, f.eks.
   • antall feil funnet
   • Antall feil funnet per side • Tid brukt på kontroll per side
   • Total gjennomgangsinnsats
   • etc.

Prosessen med gjennomgang



manager , moderator , author , reviewera , scribe

1. Uformell gjennomgang

Hensikt

Billig måte å få litt fordel på

Skjema

Par anmeldelser; f.eks. parprogrammering eller en teknisk leder som gjennomgår design og kode

Merk: Ingen formell prosess

Merk: Valgfritt kan dokumenteres

2. Gjennomgang

Formål

lære , finne feil , få forståelse , tilbakemelding

- Læring
- få forståelse
- Finne feil
- tilbakemelding

Skjema

- Møte ledet av forfatter
- kan variere i praksis fra ganske uformelle til svært formelle
- interessenter kan delta

roller manager . moderator ,authother , revirewer , scribe

3. Teknisk gjennomgang

Formål

diskutere , avgjøre , vurdere alternativer finn feil , løse tekniske problemer , inspeksjon finn feil

- diskutere
- Ta avgjørelser
- Vurdere alternativer
- Finn feil
- løse tekniske problemer
- Kontroller samsvar med spesifikasjoner og standarder

Skjema

Kan variere fra svært formell til uformell fagfellevurdering uten ledelsens deltakelse.

- Ideelt ledet av utdannet moderator
- dokumentert, definert defektdeteksjonsprosess; inkluderer kolleger og tekniske eksperter
- forberedelse før møtet
- eventuelt bruk av sjekklister, gjennomgangsrapport, liste over funn og ledelse

4. Inspeksjon

Hensikt

- Finn feil

Skjema

- Vanligvis fagfelleundersøkelse ledet av utdannet moderator (ikke forfatteren)
- Formell prosess basert på regler og sjekklister med inn- og utreisekriterier
- forberedelse før møtet
- definerte roller
- inkluderer beregninger
- inspeksjonsrapport, liste over funn

Suksessfaktorer for anmeldelser

1. Mål

Hver gjennomgang har et klart forhåndsdefinert mål

roller i reviewmanager , moderator , author , reviewer , scribe

2. Roller

De rette personene for gjennomgangsmålene er involvert.

3. Nærme seg

- Mangler som blir funnet er velkomne og uttrykt objektivt.
- Bruke passende gjennomgangsteknikker for type og nivå av programvareprodukter.
- Bruk sjekklister eller roller hvis det er hensiktsmessig for å øke effektiviteten av defektidentifikasjon.
- Ledelsen støtter en god gjennomgangsprosess (f.eks. ved å innlemme tilstrekkelig tid til gjennomgangsaktiviteter).

4. Opplæring og læring

- Det gis opplæring i gjennomgangsteknikker, spesielt de mer formelle teknikkene, som inspeksjon.
- Det legges vekt på læring og prosessforbedring

Statisk analyse av verktøy

Statisk analyse Analyse av programvareartefakter, for eksempel krav eller kode, utført uten utførelse av disse programvareutviklingsartefaktene. Statisk analyse utføres vanligvis ved hjelp av et støtteverktøy.

Mål for statisk analyse Finn feil i

- programvarens kildekode
- Programvaremodeller

Notat! Statisk analyse finner feil i stedet for feil

Statisk analyse utføres uten å faktisk utføre programvaren som undersøkes av verktøyet. Statiske analyseverktøy analyserer programkode, samt genererte utdata som HTML og XML.

Typiske feil

oppdaget av statiske analyseverktøy inkluderer:

- referere til en variabel med en udefinert verdi
- inkonsekvent grensesnitt mellom moduler og komponenter • variabler som aldri brukes
- Utilgjengelig (død) kode
- Brudd på programmeringsstandarder
- Sikkerhetsproblemer
- Syntaksbrudd på kode og programvaremodeller

tidlig oppdagelse av defekter før test utførelse

varsler mistenkelig aspekter ved kode

identifisere defekter er ikke lett ved dynamisk test

forbedrer vedlikehold av kode og design

forebygger fremtidig feil

oppdage inkonsekvenser i programmodeller

oppdage inkonsekvenser i programmodeller

Utviklere

Bruk statisk analyse før og under:

- Testing av komponenter
- Testing av integrasjon

tidlig oppdagelse av defekter
varsler mistekelig aspekter ved kode design
identifisere feil i dynamisk er ikke lett
forbedre kode og design
forebygge fremtidig feil
oppdager inkonsekvenser i progmodeller

Designere

Bruk statisk analyse under programvaremodellering

Praktisk side

Statiske analyseverktøy kan produsere et stort antall advarselsmeldinger, som må håndteres godt for å tillate mest mulig effektiv bruk av verktøyet.

Hvorfor er statisk analyse verdifull

- Tidlig oppdagelse av defekter før testutførelse.
- Tidlig advarsel om mistenkelige aspekter ved koden eller designet, ved beregning av beregninger, for eksempel et mål med høy kompleksitet.

tidlig advarsel om mistenkelige aspekter ved koden , eller design . f eks et mål med høy kompleksitet

tidlig oppdagelse av defekter før testutførelse

diskusjon finne feil avgjøre , vurdere alternativer , fiks tekniske problemer

identifisre defekter som ikke kan gjøre i dynamisk , forebygger defekter

- Identifisering av defekter som ikke er lett å finne ved dynamisk testing.
- Oppdage avhengigheter og inkonsekvenser i programvaremodeller, for eksempel koblinger.
- Forbedret vedlikehold av kode og design.
- Forebygging av defekter, hvis lærdom læres i utviklingen.

Standarder for koding

Anbefalt at eksisterende standarder bør tas i bruk for å spare mye krefter

- Sett med programmeringsregler, det vil si alltid sjekke grenser på en matrise når du bruker den
- Navnekonverteringer, for eksempel klassenavn, skal starte med stor bokstav
- Tilgangskonverteringer, for eksempel offentlige/private
- Layoutspesifikasjoner, for eksempel innrykk
- Kontrollverktøy støtter kodestandarder

Måledata for kode

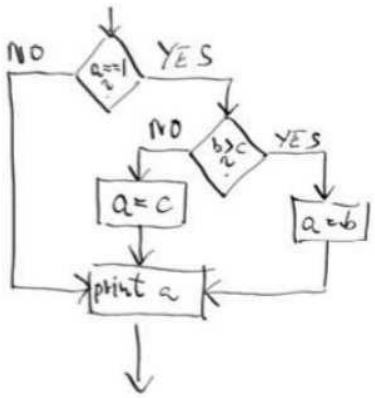
- Hyppighet av kommentarer
- Dybde på hekking
- Syklomatiske kompleksitets-/kompleksitetsmålinger

  Antall uavhengige baner gjennom et program. Syklomatisk kompleksitet er definert som: L - N + 2P, hvor

  - L = antall kanter/lenker i en graf
  - N = antall noder i en graf
  - P = antall frakoblede deler av grafen (f.eks. en kalt graf eller subrutine).

Kan måles på ulike måter, for eksempel basert på antall beslutninger i programmet (antall binære beslutninger)

```
if( a == 1 )
{   if( b > c )
      a = b;
    else
      a = c;
}
System.out.println( a );
```



Kode struktur

- Kontroller strømningsstruktur

  Rekkefølgen instruksjonene utføres i
- Struktur for dataflyt

  følger sporet til et dataelement når det åpnes og endres av koden
- Datastruktur

  Organiseringen av selve dataene , uavhengig av programmet (matrise, liste, stabel, kø, tre, graf, ...)

Verdien av statisk analyse er spesielt for

- Tidlig oppdagelse av defekt før testutførelse
- Tidlig varsling om mistenkelige aspekter ved koden, utforming og krav, ref. P4: defekt klynging

tidlig oppdagelse av defekter før test utførelse , tidlig varsling om mistenkelig anspekter ved koden

• Identifisering av defekter som ikke er lett å finne i dynamisk testing
• Forbedre vedlikeholdet av kode og design
• Forebygging av fremtidige feil

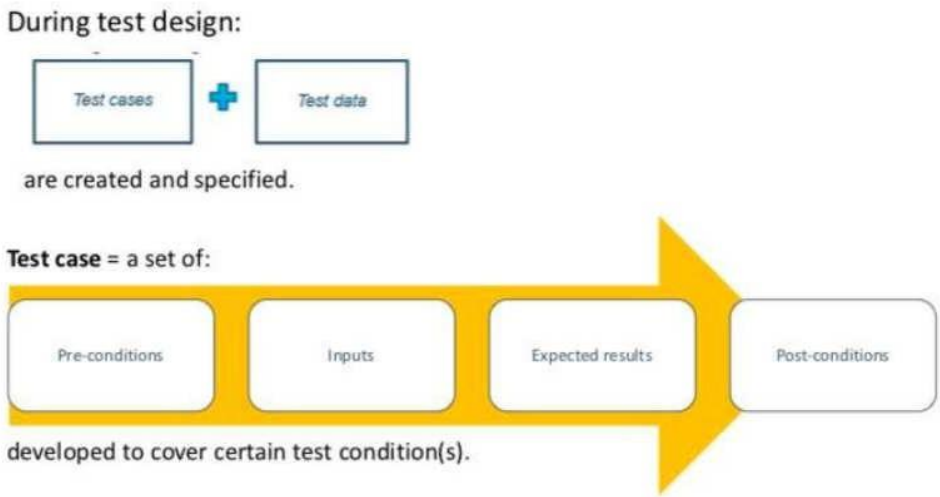
forbedre vedlikehold kode og design
forebygger fremtidige feil

identifiserer defekter man ikke finner på dynamisk testing , forbedrer kode og design forebygger fremtiidig feil

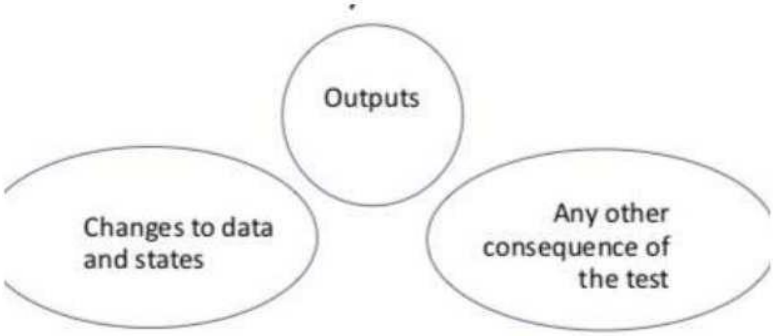
Kapittel 4

Testdesignprosessen kan gjøres på forskjellige måter, fra veldig uformell (lite eller ingen dokumentasjon), til veldig formell.
Formalitetsnivået avhenger av konteksten for testingen, inkludert:

Test utvikling:
1. Test analyse
Testgrunnlagsdokumentasjonen analyseres for å bestemme hva som skal testes, det vil si for å identifisere testforholdene.
Testbetingelse (Def.) = et element eller en hendelse som kan verifiseres av ett eller flere testtilfeller Eksempler
• En funksjon
• En transaksjon
• En egenskap
• Andre strukturelle elementer (menyer på nettsider
osv.) Testmuligheter:
«Kast et bredt nett!»
Første; identifisere så mange testforhold som mulig For det andre; Velg hvilken du vil utvikle mer detaljert
Vi kan ikke teste alt (P2). Vi må velge en delmengde av alle mulige tester, men denne delmengden må ha stor sannsynlighet for å finne de fleste feilene i systemet.
Vi trenger en passende testdesignteknikk for å veilede vårt valg og for å prioritere testbetingelsene.
2. Test design



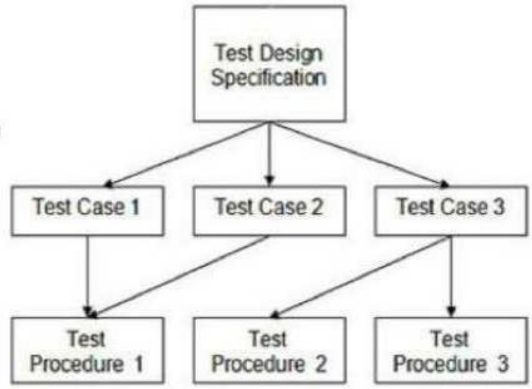
Test orakel:
For å vite hva systemet skal gjøre, må vi ha en kilde til informasjon om riktig oppførsel av systemet – et orakel.
Forventede resultater inkluderer:



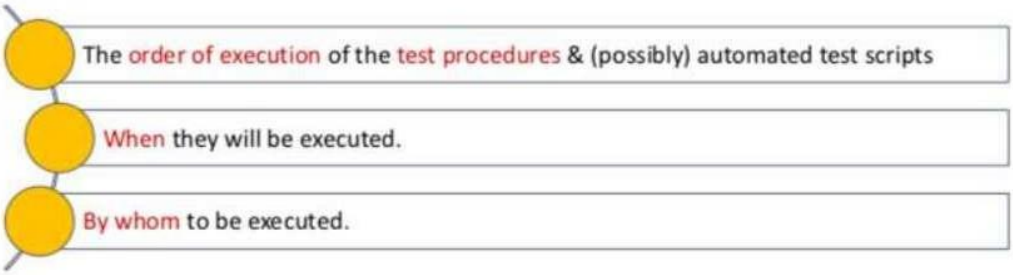
Hvis forventede resultater ikke er definert, kan et plausibelt, men feilaktig resultat tolkes som det riktige.
Forventede resultater bør ideelt sett defineres før testutførelse.

3. Test implementering
Under testimplementeringen er testtilfellene organisert
i testprosedyrene:

En manuell testprosedyre
spesifiserer rekkefølgen av handlinger som skal utføres for å utføre en test.
En automatisert testprosedyre (testskript)
Hvis tester kjøres ved hjelp av et testutførelsesverktøy, angis handlingssekvensen i et testskript.



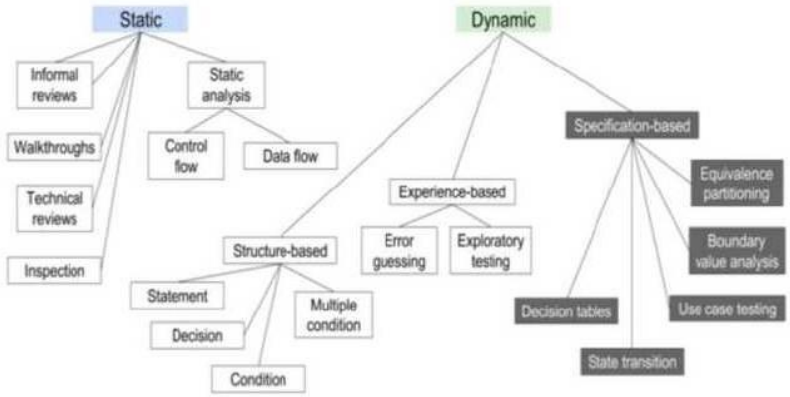
Tidsplanen for testutførelse definerer:



Testutførelsesplanen vil ta hensyn til faktorer som:
- risiki
- Regresjonstester
- prioritering
- tekniske og logiske avhengigheter

Å skrive testprosedyren er en annen mulighet til å prioritere testene, for å sikre at den beste testingen gjøres på den tiden som er tilgjengelig.
En god tommelfingerregel er "Finn de skumle tingene først". Definisjonen av hva som er "skummelt" avhenger imidlertid av
virksomheten, systemet eller prosjektet og om risikoen ved

prosjektet. Kategorier av testdesignteknikker:



Black-box-testing:

- Vi bruker modeller (formelle eller uformelle) for å spesifisere problemet som skal løses, samt programvaren eller dens komponenter.
- Vi utleder systematisk testtilfellene fra disse modellene.
- Testcasene er avledet fra informasjon om hvordan programvaren er bygd opp, for eksempel kode og design.
- For de eksisterende testtilfellene kan vi måle testdekningen til programvaren.
- Ytterligere testtilfeller kan utledes systematisk for å øke testdekningen.
- Testtilfellene er avledet fra kunnskapen og erfaringen til mennesker:
    - Kunnskap om testere, utviklere, brukere og andre interessenter om programvaren, dens bruk og dens miljø.
    - Kunnskap om sannsynlige defekter og deres utbredelse.

Partisjonering av ekvivalens

• Den grunnleggende ideen er å dele et sett med testbetingelser inn i partisjoner (delsett) der elementene i hver partisjon som kan betraktes som like.

• Det er viktig at de forskjellige skilleveggene ikke har felles elementer.

• Vi trenger bare å teste en tilstand fra hver partisjon, fordi alle betingelsene i samme partisjon vil bli behandlet på samme måte av programvaren.

Eksempel: se side 81-83 i boka

Teknikk

Innganger/utganger/interne verdier av programvaren er delt inn i grupper som forventes å vise lignende oppførsel.

Ekvivalenspartisjoner kan finnes for både gyldige data og ugyldige data, det vil si verdier som skal være avslått.

Notater

• Tester kan utformes for å dekke mer enn én partisjon.

• Ekvivalenspartisjonering gjelder på alle testnivåer.

• Ekvivalenspartisjonering som en teknikk kan brukes for å oppnå inngangs- og utgangsdekning.

Analyse av grenseverdi

Det er mer sannsynlig at virkemåten på kanten av hver ekvivalenspartisjon er feil enn oppførselen i partisjonen.

Grense er et område der testing sannsynligvis vil gi defekter.

Grenseanalyse

Analyse ved kanten av hver ekvivalenspartisjon. Hvorfor? For der er det mer sannsynlig at resultatene er feil.

Maksimums- og minimumsverdiene til en partisjon er grenseverdiene.

Gyldig og ugyldig grense

- En grenseverdi for en gyldig partisjon er en gyldig grenseverdi.
- En grenseverdi for en ugyldig partisjon er en ugyldig grenseverdi.
- Testene kan utformes slik at de dekker både gyldige og ugyldige grenseverdier.

Notater

- Grenseverdianalyse kan brukes på alle testnivåer.
- Den er relativt enkel å påføre og evnen til å finne feil er høy.
- Detaljerte spesifikasjoner er nyttige.
- Denne teknikken blir ofte betraktet som en utvidelse av ekvivalenspartisjonering.
- Grenseverdier brukes for valg av testdata.

Hvorfor gjøre både ekvivalenspartisjonering og grenseverdianalyse?

Grenseverdier er vanligvis ekstremverdier. For å få tillit til systemet ønsker vi også å teste det under normale omstendigheter.

Testing av beslutningstabell

Beslutningstabeller er en god måte

- for å registrere systemkrav som inneholder logiske betingelser
- for å dokumentere intern systemdesign
- å registrere komplekse forretningsregler som et system skal implementere

Når du oppretter beslutningstabeller, analyseres spesifikasjonen, og handlingene til systemet identifiseres. Hvis inndatabetingelsene og handlingene er angitt på en måte der de enten er sanne eller usanne (boolsk), kan beslutningstabeller være nyttige.

Beslutningstabellen inneholder de utløsende betingelsene, det vil si alle kombinasjoner av sant og usant for alle inndata

betingelser, og de resulterende handlingene for hver kombinasjon av betingelser.

Testing av tilstandsovergang

Et system kan være i et begrenset antall forskjellige tilstander. Disse aspektene ved systemet kan beskrives som en 'endelig tilstandsmaskin'; et tilstandsdiagram.

Ethvert system der du får en annen utgang for samme inngang, avhengig av hva som har skjedd før, er et endelig tilstandssystem.

Overgangen fra en tilstand til en annen bestemmes av reglene til 'maskinen'.

State transition testing

System can be in a finite number of different states

- Elements of state transition models

States → The SW may occupy

E.g. open / closed, active / inactive

Transitions → From one state to another

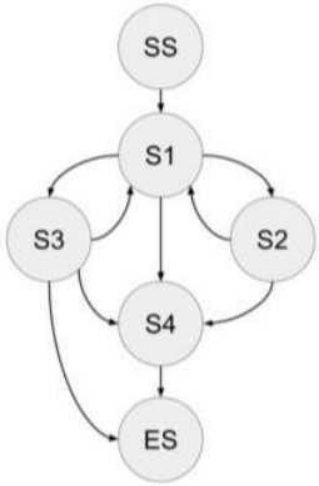
Not all transitions are allowed

Events → Causing state transitions

E.g. closing a file, withdrawing money

Actions → Actions resulting from transitions

E.g. error message



Hvorfor testing av tilstandsoverganger?

Fordi et system kan vise en annen respons avhengig av gjeldende forhold eller tidligere historie. Testing av tilstandsovergang lar testeren se:

- programvaren når det gjelder dens tilstander
- Overganger mellom tilstander
- inndata eller hendelser som utløser tilstandsendringer (overganger)
- handlingene som kan følge av disse overgangene

Tester kan utformes

- for å dekke en typisk sekvens av tilstander
- for å utøve spesifikke sekvenser av overganger
- for å dekke alle stater
- å utøve hver overgang
- Slik tester du ugyldige overganger

Tilstandsovergangstesting er mye brukt innen programvareindustrien og teknisk automatisering generelt.

Testing av brukstilfeller

Brukstilfelle - beskriver interaksjoner mellom aktører (brukere og systemet), som gir et resultat av verdi for en systembruker.

Use case testing

- Identify test cases that exercise the whole system
  - Transaction by transaction basis
  - From start to finish
- Describes interactions between actor and system
  - Achieve a specific task
  - Produce something of value to the user
- Defined in terms of the actor, not the system
  - Describes process flows through a system
    - Based on its actual use
  - Can uncover integration defects

Use case name	<name>	
Actor(s)	<actor1>, ...	
Pre-conditions	<cond1>, ...	
Post-conditions	<cond1>, ...	
Main Success Scenario	Step	Description
	1	A: <action>
	2	S: <response>
	3	A: <action>
	4	S: <response>
	...	...
Extensions	Step	Description
	S.X	<cause> S: <response>
	S.Y	<cause> S: <response>

• Brukstilfeller er svært nyttige for å utforme aksepttester med kunde-/brukermedvirkning.
• Brukstilfeller beskriver «prosessflytene» gjennom et system basert på den faktiske sannsynlige bruken.
• Testtilfellene avledet fra brukstilfeller er mest nyttige for å avdekke defekter i prosessflytene under reell bruk av systemet.
• De hjelper også med å avdekke integrasjonsfeil forårsaket av integrasjon og interferens av forskjellige komponenter, som individuell testing ikke ville se.
• Utforming av testtilfeller fra brukstilfeller kan kombineres med andre spesifikasjonsbaserte testteknikker.

Strukturbasert testing.
White-box-teknikker.
Strukturbaserte teknikker tjener to formål:
Måling av testdekning
Vi kan vurdere hvor mye testing som utføres av tester avledet fra for eksempel spesifikasjonsbasert teknikk for å vurdere dekning.
Design av strukturelle testtilfeller
Vi kan generere flere testtilfeller med sikte på å øke testdekningen.

$$\text{Coverage} = \frac{\text{Number of coverage items exercised}}{\text{Total number of coverage items}} \times 100\%$$

En dekningspost er det vi har vært i stand til å telle og se om en test har utøvd eller brukt denne posten.
NB! 100% dekning betyr ikke at 100% testet!

Strukturbasert testing (white-box) er basert på en identifisert struktur av programvaren.
Komponentnivå - strukturen er selve koden: - utsagn, beslutninger/grener. Integrasjonsnivå - strukturen kan være et kalletre (et diagram der moduler kaller andre moduler).

Systemnivå - strukturen kan være en menystruktur, forretningsprosess eller nettsidestruktur.

Hvordan måle dekning?

Trinnene som vanligvis tas for å måle dekning er nyttige for å forstå de relative fordelene ved hver teknikk:

1. Bestem deg for det strukturelle elementet som brukes, det vil si dekningselementene som skal telles.
2. Tell strukturelle elementer eller gjenstander.
3. Instrumenter koden.
4. Kjør testene som dekningsmålingen er nødvendig for.
5. Ved å bruke utdataene fra instrumenteringen, bestemme prosentandelen av elementer eller gjenstander som utøves.

I komponenttesting

Uttalelsesdekning er prosentandelen av kjørbare uttalelser som har blitt utøvd av en testcase-suite.

Utsagntestingsteknikken utleder testtilfeller for å utføre spesifikke utsagn, normalt for å øke utsagndekningen.

$$\text{Statement coverage} = \frac{\text{Number of statements exercised}}{\text{Total number of statements}} \times 100\ \%$$

Beslutningsdekning - er vurderingen av prosentandelen av beslutningsutfall (f.eks. og falske alternativer for en IF-erklæring) som har blitt utøvd av en testcase-suite.

$$\frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100\ \%$$

Beslutningsdekning er sterkere enn uttalelsesdekning. 100 % beslutningsdekning garanterer 100 % kontoutsagnsdekning

Overgangsdekningen er sterkere enn statlig dekning

100 % overgangsdekning garanterer 100 % statlig dekning Ikke omvendt!

Opplevelsesbasert testing

- Tester er avledet fra testerens ferdigheter og intuisjon og deres erfaring med lignende applikasjoner og teknologier.
- Når den brukes til å forsterke systematiske teknikker, kan erfaringsbasert testing være nyttig i identifisere spesielle tester som ikke lett fanges opp av formelle teknikker, spesielt når de brukes etter mer formelle tilnærminger.
- Kan gi svært varierende grad av effektivitet, avhengig av testerens erfaring.

Feil gjetting = en ofte brukt erfaringsbasert teknikk. Vanligvis forutser testere defekter basert på erfaring.

En strukturert tilnærming til feilgjetteteknikken er å liste opp en liste over mulige feil og for å designe tester som angriper disse feilene.

tester bruker sin erfaring og instuisjon i programmer og system , tester lærer om systeme når de tester . brukes for å finne og avdekke feil

Denne systematiske tilnærmingen kalles feilangrep.

Utforskende testing = samtidig testdesign, testutførelse, testlogging og læring, basert på et testcharter som inneholder testmål, og utført innenfor tidsbokser.

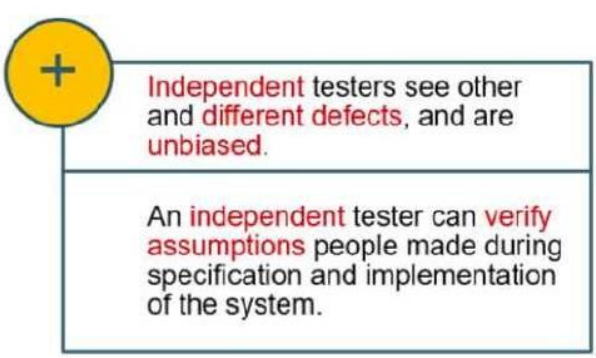
Det er mest nyttig ...

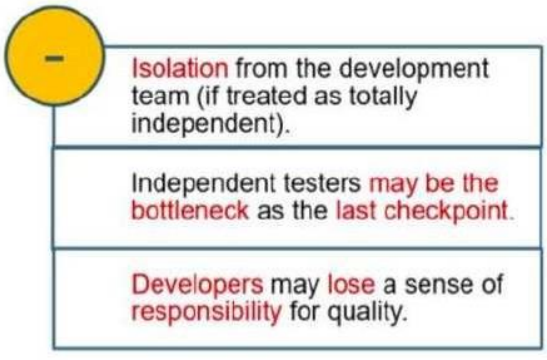
- der det er få eller utilstrekkelige spesifikasjoner
- under sterkt tidspress
- for å utfylle andre, mer formelle tester
- Det kan tjene til å sikre at de mest alvorlige feilene blir funnet.

Kapittel 5

- Testing er en vurdering av kvalitet
- Skill testerne fra utviklerne
- Effektiviteten av å finne defekter ved testing og vurderinger kan forbedres ved å bruke uavhengige testere.
- Alternativer for uavhengighet er:

- For store, komplekse eller sikkerhetskritiske prosjekter er det vanligvis best å ha flere nivåer av testing, med noen eller alle nivåene utført av uavhengige testere.
- Utviklingspersonell kan delta i testing, spesielt på de lavere nivåene





Notat

Testoppgaver kan utføres av personer i en bestemt testrolle, eller kan utføres av noen i en annen rolle, for eksempel:

- Prosjektleder
- Kvalitetssjef
- utvikler
- Forretnings- og domeneekspert
- infrastruktur eller IT-drift

(dette kan være både bra og dårlig)

Oppgaver for testleder og tester:

Det er stor variasjon i rollene som personer i testteamet spiller. De to vanligste rollene er

- *Testleder*
- *tester*

Testleder = testleder/testkoordinator.

Testlederen planlegger, overvåker og kontrollerer testaktivitetene og oppgavene. Testeren

Testeren gjennomgår og bidrar til testplanlegging, analyserer, designer, forbereder, gjennomfører og gjennomfører tester.

Tasks of the test leader

Coordination	of the test strategy and plan with project managers
Plan the tests	Understanding the test objectives and risks – including: • selecting test approaches • estimating the time, effort and cost of testing • acquiring resources • defining test levels, cycles • planning incident management
Test specifications, preparation and execution	Initiate the specification, preparation, implementation and execution of tests • monitor the test results • check the exit criteria.
Adapt planning	based on test results and progress and take any action to compensate for problems.
Manage test configuration	Set up adequate configuration management of testware for traceability.
Introduce metrics	for measuring test progress and evaluating the quality of testing & product.
Automation of tests	Decide what should be automated, to what degree, and how.
Select test tools	Select tools to support testing and organize trainings for tool users.
Test environment	Decide about the implementation of the test environment.
Test summary reports	Write test summary reports based on the information gathered during testing.

Tasks of the tester

Test plans	Review and contribute to test plans
Requirements and specifications	Analyze, review and assess user requirements, specifications and models for testability.
Test specifications	Create test specifications
Test environment	Set up the test environment (often coordinating with system administration and network management).
Test data	Prepare and acquire test data.
Testing process	Implement tests on all test levels, execute and log the tests, evaluate the results and document the deviations from expected results.
Test tools	Use test tools (for administration, management or monitoring) as required.
Test automation	Automate tests (may be supported by a developer or a test automation expert).
Other metrics	Measure performance of components and systems (if applicable).
Help the others	Review tests developed by others

Personer som er involvert i testing trenger

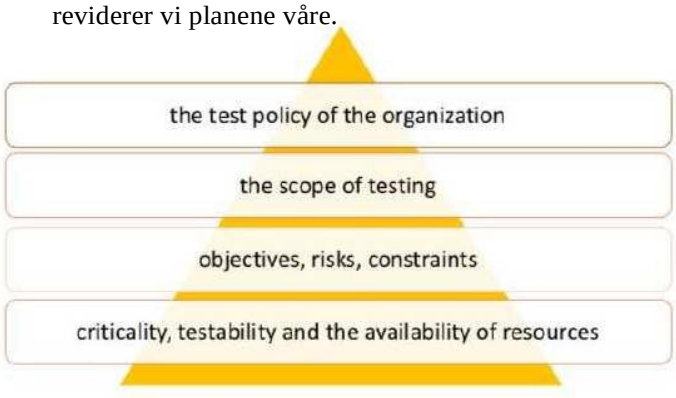
- Applikasjons- eller forretningsdomene: En tester må forstå den tiltenkte oppførselen og problemet systemet vil løse for å oppdage feil.
- Teknologi: En tester må være klar over problemer, begrensninger og evner til den valgte implementeringsteknologi, for å lokalisere problemer og funksjonene og funksjonene som "sannsynligvis vil mislykkes".
- Testing: En tester må kjenne til testtemaene som er diskutert i denne boken - for å kunne utføre testoppgavene som er tildelt.
- Grunnleggende faglige og sosiale kvalifikasjoner som • lese- og skriveferdigheter
  - evne til å utarbeide og levere skriftlige og muntlige rapporter
  - evnen til å kommunisere effektivt
  - ...

Testplanlegging og estimering

En testplan er prosjektplanen for testarbeidet som skal gjøres.

leder planlegger , gjennomfører , lager test data , setter opp miljø , bruker verktøy for å teste , analyse , design , forbereder miljø

- Å skrive en testplan tvinger oss til å konfrontere utfordringene som venter oss og fokusere vår tenkning på viktige temaer.
- Testplanleggingsprosessen og selve planen fungerer som redskaper for å kommunisere med andre medlemmer av prosjektteamet, testere, kolleger, ledere og andre interessenter.
- Testplanen hjelper oss også med å håndtere endringer. Etter hvert som vi samler inn mer informasjon, reviderer vi planene våre.



*Outlines of test planning documents are covered by the 'Standard for

Planlegging kan dokumenteres i:

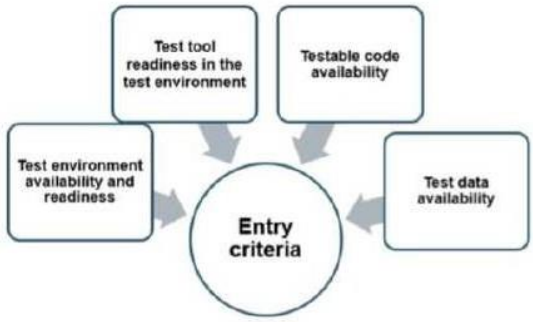
- en prosjekt- eller hovedtestplan
- og i separate testplaner for testnivåer, for eksempel integrasjonstesting, systemtesting og aksepttesting.

plan definere mål , strategier , scope , exit og entry kriterieer , stategier ressursallokering risiko , asvar og roller

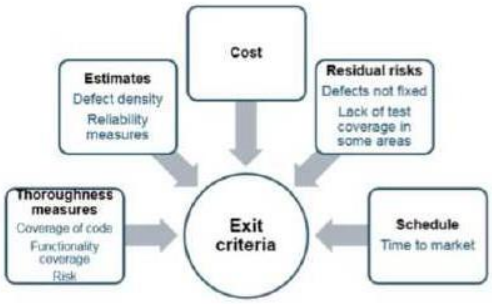
Testplanlegging er en kontinuerlig aktivitet og utføres i alle livssyklusprosesser og aktiviteter. Tilbakemeldinger fra testaktiviteter brukes til å gjenkjenne endrede risikoer slik at planleggingen

Test planning activities	
Scope and risk	Determining the scope and risks of testing
Objectives	Identifying the objectives of testing.
Overall approach	Defining the overall approach of testing, including: • the definition of the test levels • entry and exit criteria.
Test activities	Integrating and coordinating the testing activities into the software life cycle activities: • acquisition and supply • development • operation • maintenance
Strategy	Making decisions about: • what to test • what roles will perform the test activities • how the test activities should be done • and how the test results will be evaluated.
Schedule	Scheduling test analysis and design activities. Scheduling test implementation, execution and evaluation
Resources	Assigning resources for the different activities defined.
Metrics	Selecting metrics for monitoring and controlling test preparation and execution, defect resolution and risk issues.

Entry criteria defines when to start testing.



Exit criteria is to define when to stop testing, such as at the end of a test level, end of project or when a set of tests has a specific goal.



Testestimering

Testarbeidet er vanligvis et delprosjekt innenfor det større prosjektet. Grunnleggende estimeringsteknikker kan tilpasses for testing

Vi kan dele opp et testprosjekt i faser • planlegging og kontroll

- Analyse og design
- Implementering og gjennomføring
- Evaluering av exit-kriterier og rapportering
- test stenging

plan kontroll
analyse design
implementasjon utførelse
evaluere exit og rapportering
ferdigstilling

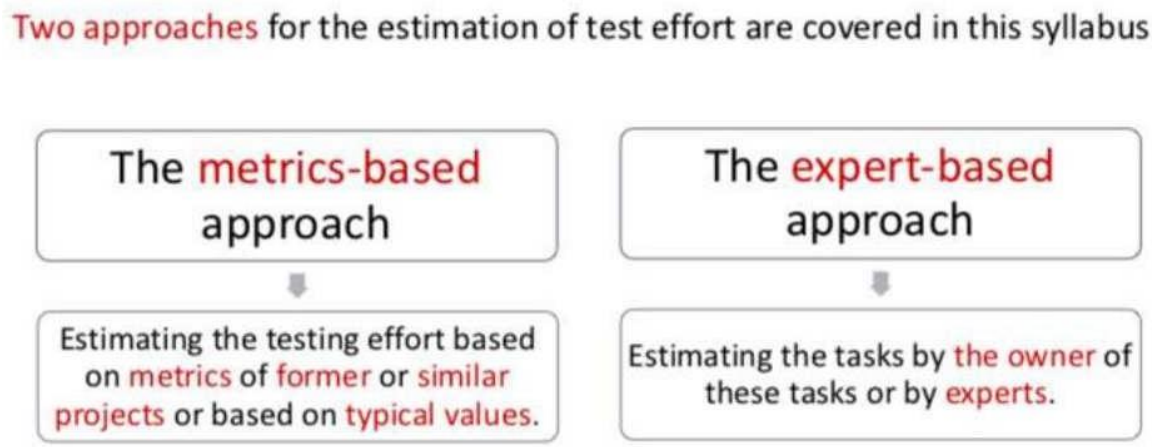
Innenfor hver fase identifiserer vi aktiviteter og innenfor hver aktivitet identifiserer vi oppgaver og kanskje deloppgaver. For å identifisere aktivitetene og oppgavene jobber vi både

- fremover
- Bakover

For å sikre nøyaktigheten av estimatet, sørg for at du deler arbeidet inn i oppgaver som er kortvarige, for eksempel én til tre dager.

beskrive inffo arbeidsprossene , hva som ple gjort

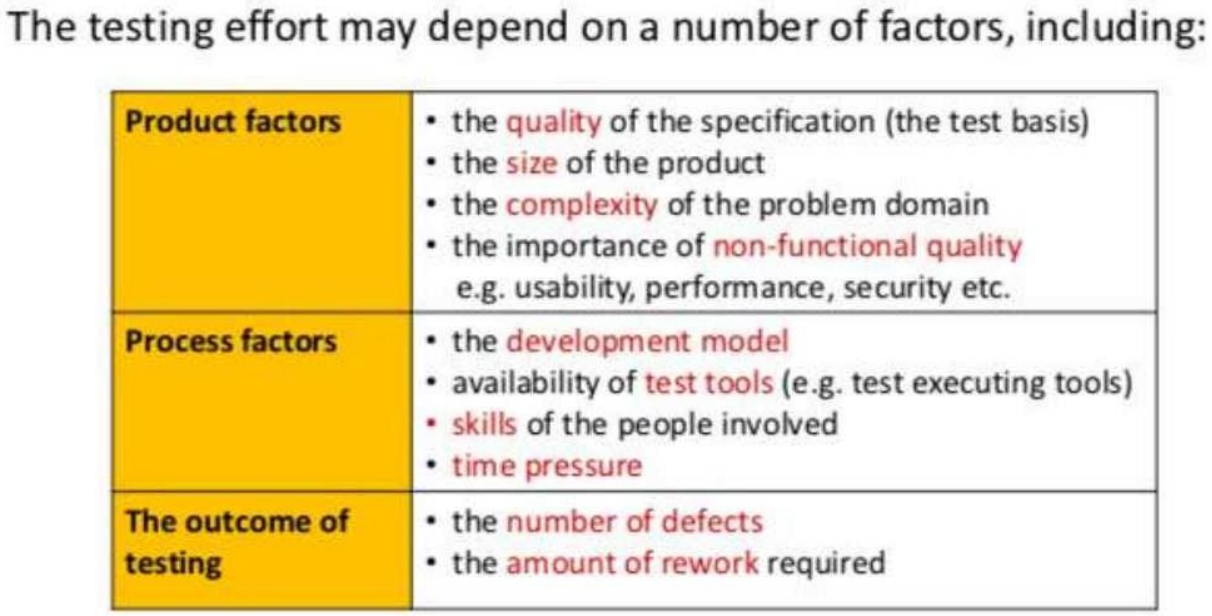
Hvis de er mye lengre – la oss si to uker – vil det være en risiko for at lange og komplekse deloppgaver «skjules» i den større oppgaven.



En god løsning er å kombinere de to strategiene:

- Lag først arbeidsnedbrytningsstrukturen og et detaljert nedenfra-og-opp-estimat.
- Andre; Vi bruker deretter modeller og tommelfingerregler for å sjekke og justere estimatet nedenfra og opp og ovenfra og ned ved hjelp av tidligere historikk.

Denne tilnærmingen har en tendens til å skape et estimat som er både mer nøyaktig og mer forsvarlig enn begge teknikkene alene.



Test tilnærminger og strategier:
Testtilnærmingen er implementeringen av teststrategien for et spesifikt prosjekt.
Siden testtilnærmingen er spesifikk for et prosjekt, bør tilnærmingen dokumenteres i testplanen. En måte å klassifisere testtilnærminger eller strategier på er basert på tidspunktet da hoveddelen av testdesignarbeidet begynner:
Forebyggende tilnærminger

- Testene utformes så tidlig som mulig

forebyggene utformes før systemet er bygget , reaktiv utformes etter at systemet er bygd

Reaktive tilnærminger

- Testdesign kommer etter at programvaren eller systemet er produsert

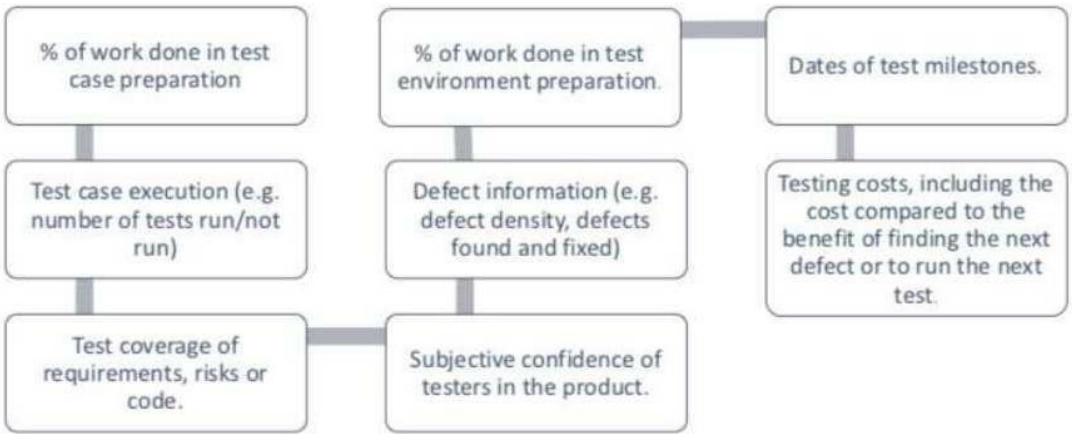
Analytical approaches	e.g. risk-based testing - testing is directed to areas of greatest risk, requirement-based testing
Model-based approaches	e.g. testing using statistical information about failure rates (such as reliability growth models)
Methodical approaches	e.g. failure-based (including error guessing and fault-attacks), experienced-based, check-list based, and quality characteristic based.
Process- or standard-compliant approaches	e.g. specified by industry-specific standards or the various agile methodologies.
Dynamic and heuristic approaches	e.g. exploratory testing, execution & evaluation are concurrent tasks.
Consultative approaches	e.g. test coverage is evaluated by domain experts outside the test team.
Regression-averse approaches	include reuse of existing test material, extensive automation of functional regression tests.

Valget av testtilnærminger og strategier er en kraftig faktor for suksessen til innsatsen og nøyaktigheten til testplanene og estimatene. Når vi velger teststrategier er det mange faktorer å vurdere:

- Risiki
- Ferdigheter
- Mål
- Regelverk
- Produkt
- Forretning

Overvåking av testfremdrift

- Hensikten med testovervåking er å gi tilbakemelding og synlighet om testaktiviteter.
- Informasjon som skal overvåkes kan samles inn manuelt eller automatisk og kan brukes til å måle utgangskriterier, for eksempel testdekning.
- Beregninger kan også brukes til å vurdere fremdriften mot den planlagte tidsplanen og budsjettet.



IEEE 829
829 Standard for programvare og systemtestdokumentasjon Testlogg - mal

Test-rapportering
Testrapportering er opptatt av å oppsummere informasjon om testarbeidet, inkludert:

- Hva skjedde i testperioden? (f.eks. datoer da utreisekriteriene ble oppfylt)

- Analyserte beregninger for å støtte beslutninger om fremtidige handlinger (f.eks. den økonomiske fordelen av fortsatt testing)

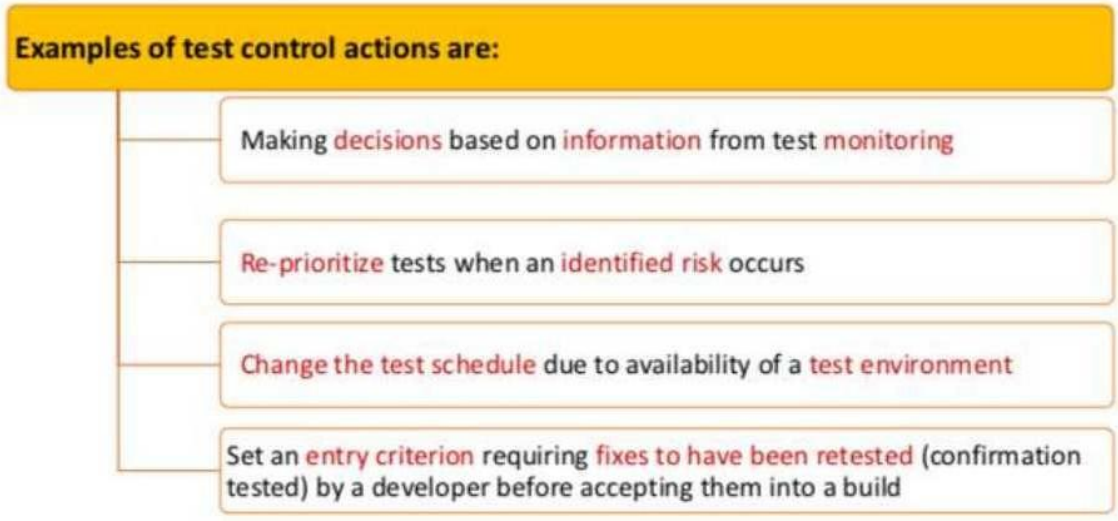
Beregninger samles inn på slutten av et testnivå for å vurdere:

- Tilstrekkeligheten til testmålene for det testnivået
- Testtilnærmingenes tilstrekkelighet med hensyn til deres mål
- Effektiviteten av testingen med hensyn til målene

IEEE 829
829 Standard for programvare og systemtestdokumentasjon Testsammendragsrapport – mal
Test kontroll
Prosjekter går ikke alltid etter planen. Ulike faktorer kan føre til avvik:

- Nye risikoer
- Nye behov
- Testfunn
- Eksterne arrangementer
- Problemer med testmiljøet

Testkontroll beskriver eventuelle veiledende eller korrigerende tiltak som er iverksatt som et resultat av informasjon og beregninger som er samlet inn og rapportert.



Administrasjon av konfigurasjoner
Hensikten med konfigurasjonsadministrasjon er å etablere og opprettholde integriteten til programvaren og relaterte produkter gjennom prosjektets og produktets livssyklus.
Konfigurasjonsstyringen skal sikre at alle testvarer er

- Identifisert
- versjonskontrollert
- spores for endringer

slik at sporbarheten kan opprettholdes gjennom hele testprosessen.

Alle identifiserte dokumenter og programvareelementer bør refereres entydig i testdokumentasjonen. Konfigurasjonsadministrasjon hjelper deg med å identifisere (og reprodusere) unikt

Det testede elementet □*Testdokumenter* □*Testene* □*Testselen*

ignorer , redusererr , forebygge beerdskap forflyrning

Prosedyrer og verktøy for konfigurasjonsadministrasjon bør velges i prosjektplanleggingsfasen.

Risiko og testing

• Risiko er muligheten for et negativt eller uønsket utfall, mulige problemer som kan sette målene til prosjektets interessenter i fare.
• Risikoer er knyttet til produktet/prosjektet
• Risikoanalyse og risikostyring kan hjelpe oss med å stake ut en kurs for solid testing. Risikonivået bestemmes av:

- Sannsynligheten for at en uønsket hendelse inntreffer
- Virkningen (skaden som følge av den hendelsen)

For eventuelle risikoer har du fire muligheter: • Reduser • Beredskap • Forflytning
• Ignorere

Prosjekt risikoer

- Logistikk- eller produktkvalitetsproblemer som blokkerer tester
- Test elementer som ikke kan installeres i testmiljøet
- Overdreven endring av produktet som ugyldiggjør testresultater eller krever oppdateringer av testtilfeller, forventede resultater og miljøer
- Utilstrekkelige eller urealistiske testmiljøer som gir misvisende resultater

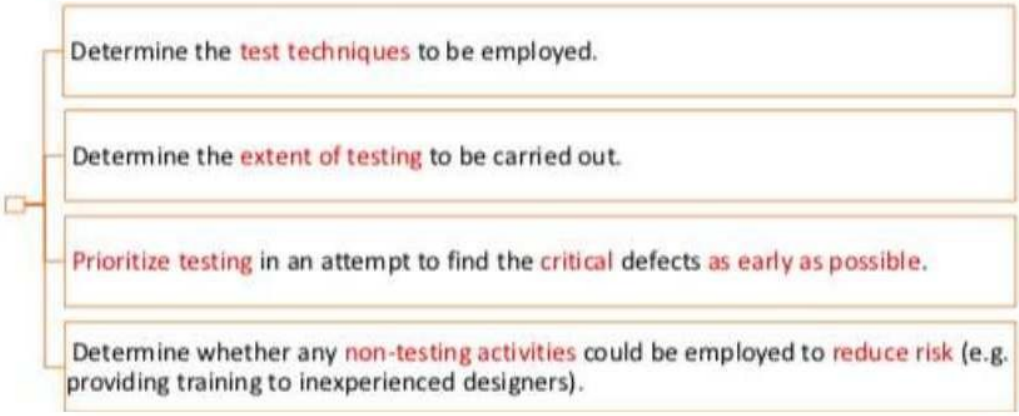
Prosjektrisiko = risikoene som omgir prosjektets evne til å levere sine mål, for eksempel:

Organizational factors:	Technical issues:	Supplier issues:
• skill and staff shortages • personal and training issues • problems with testers communicating their needs and test results • improper attitude toward testing (i.e. not appreciating the value of finding defects during testing).	• problems in defining the right requirements • the extent that requirements can be met given existing constraints • the quality of the design, code and tests.	• failure of a third party • contractual issues.

Produktrisiko = er en risiko som er direkte relatert til testobjektet Produktrisiko er mulighete n for at systemet eller programvare kan mislykkes i å tilfredsstille rimelige kunde-, bruker- eller interessentforventninger. Produktrisikoer = Potensielle feilområder i programvaren. Risikoer brukes til å bestemme hvor du skal begynne å teste og hvor du skal teste mer.

Risikobasert tilnærming

Testing brukes for å redusere risikoen for at en bivirkning oppstår, redusere virkningen av en bivirkningså tidlig som mulig. I en risikobasert tilnærming kan de identifiserte risikoene brukes til å:



Risikoanalyse bør starte så tidlig som mulig:

- Identifisere risikoelementene
- Bestem sannsynligheten og virkningen for hvert element
- Bruk en vurderingsskala (1 – 10) til å klassifisere risikonivået for hver vare
- Prioriter risikopostene i henhold til deres ratingverdier

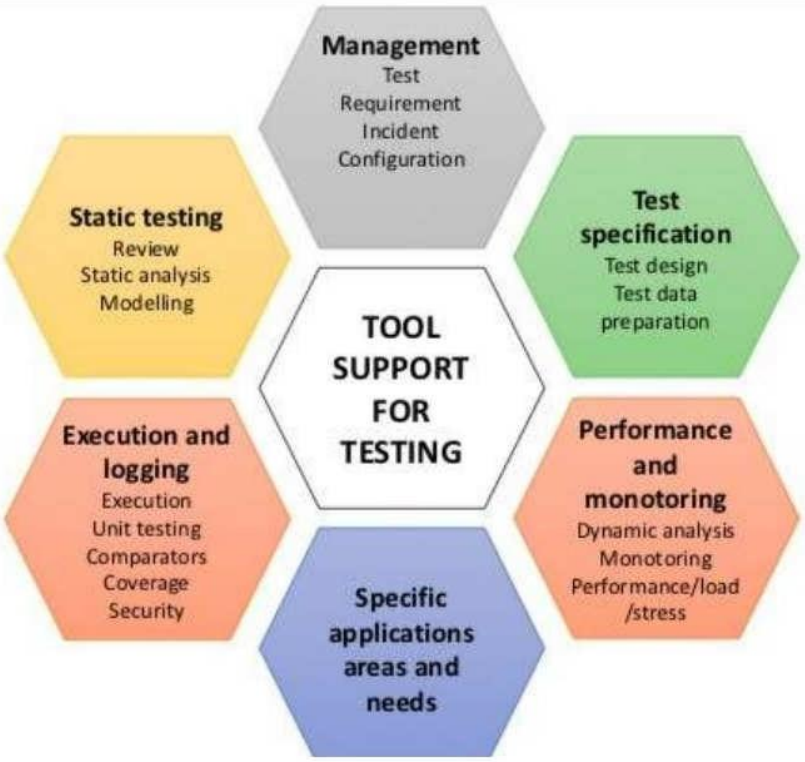
1. Analyser risiko tidlig i prosjektet.
2. Du bør håndtere risikoer på riktig måte, basert på sannsynlighet og innvirkning, men ikke forveksle innvirkning med sannsynlighet eller omvendt.
3. Målet med risikobasert testing bør ikke være - kan i praksis ikke være - et risikofritt prosjekt.
4. Beste praksis innen risikostyring for å oppnå et prosjektresultat som balanserer risiko med kvalitet, funksjoner, budsjett og tidsplan.

Håndtering av hendelser
Hendelse - Avvik mellom faktiske og forventede testresultater.
Hendelseshåndtering - Prosessen med å gjenkjenne, undersøke, iverksette tiltak og avhende hendelser.
Hendelsesrapport - Et rapportdokument som rapporterer om enhver hendelse som skjedde, for eksempel under testing, som krever undersøkelse.
Hva skal med i en hendelsesrapport?

- En beskrivelse av en situasjon, atferd eller hendelse som skjedde.
- En eller to skjermer – med informasjon samlet inn av et feilsporingsverktøy.
- En beskrivelse av trinnene som er gjort for å reprodusere og isolere hendelsen.
- Virkningen av problemet.
- Klassifiseringsinformasjon (dvs. mangelens omfang, alvorlighetsgrad og prioritet). Et prioritetsnivå, tildelt av testlederne
- Risikoene, kostnadene, mulighetene og fordelene forbundet med å fikse eller ikke fikse defekt tildelt av prosjektgruppen eller en komité.
- Årsaken, fanget av programmereren, - introduksjonsfasen - fasen av fjerning
- Konklusjoner og anbefalinger fanget opp av ledere, programmerere eller andre
- Gjennom hele livssyklusen til hendelsesrapporten bør defektsporingssystemet tillate hver person som jobber med hendelsesrapporten å legge inn status- og historikkinformasjon.

Kapittel 6

Typer testverktøy



Typer: • Verktøy som brukes direkte i testing (f.eks. testutførelsesverktøy, datagenereringsverktøy, verktøy for sammenligning av resultater)

• Verktøy som hjelper til med å administrere testprosessen (f.eks. testresultater, krav, hendelser, defekter) og for å overvåke og rapportere testutførelsen

• Verktøy som brukes i utforskning (f.eks. verktøy som overvåker filaktiviteten for et program)

• Ethvert verktøy som hjelper til med testing

Formål: • forbedre effektiviteten til testaktivitetene (f.eks.: ved å automatisere repeterende oppgaver)

• Automatiser aktiviteter som krever betydelige ressurser når de gjøres manuelt (f.eks. statisk testing)

• Automatiser aktiviteter som ikke kan gjøres manuelt (f.eks. storskala ytelsestesting av klient-server-applikasjoner)

• øke påliteligheten til testing (ved å automatisere store datasammenligninger eller simulere kompleks oppførsel)

Notater

• Noen typer testverktøy kan være påtrengende - selve verktøyet kan påvirke utfallet av testen. (f.eks. kan tidsmålinger være forskjellige avhengig av hvordan du måler det med forskjellige ytelsesverktøy).

• Konsekvensen av påtrengende verktøy kalles sondeeffekten.

• Noen verktøy tilbyr støtte som er mer passende for utviklere. Slike verktøy er merket med "(D)" i dette kapittelet.

Bompengestøtte for administrasjon

Egenskaper

- Støtte til ledelse av tester og testaktiviteter.
- Støtte for sporbarhet av tester, testresultater og hendelser til kildedokumenter, for eksempel kravspesifikasjoner.
- Generasjon av fremgang Rapporter.
- Logging prøve Resultater. Monotoring
- Tilby informasjon om beregninger relatert

Incident management tools	store and manage incident reports
	support management of incident reports

Requirements management tools	store requirements
	check for consistency and undefined (missing) requirements
	allow prioritization
	enable individual tests to be traceable to requirements

til tes tene.

Configuration management tools	are necessary to keep track of different versions and builds of the SW and tests
	are particularly useful when developing on more than one configuration of the HW/SW environment

Verktøystøtte for statisk testing

Tools for static testing

Tools that aid in improving the code / work product, without executing it

Categories

Review tools

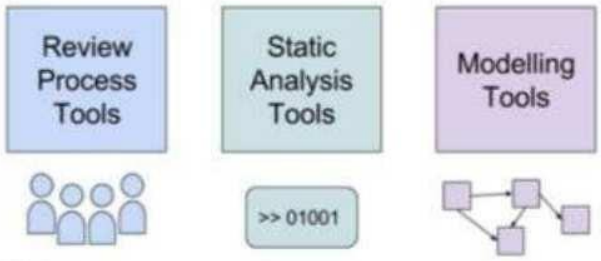
Supports the review process

Static analysis tools

Supports code examination

Modelling tools

Validate models of system / software



Review process tools

Common reference for the review processes conducted

Keep track of all the information from the review process

Store and communicate review comments, report on defects and effort

Monitoring review status → Passed, passed with corrections, requires re-review

When to use?

Suitable for more formal review processes

Geographically dispersed teams

Static analysis tools (D)	Major purpose: - The enforcement of coding standards. - The analysis of structures and dependencies (e.g. linked webpages) - Aiding in understanding the code.
	support developers, testers and quality assurers in finding defects before dynamic testing.
	Static analysis tools can calculate metrics from the code (e.g. complexity), which can give valuable information for planning or risk analysis.