Certified Tester
Pensum for Foundation-niveauet

Frigivet
Version 2011

International Software Testing Qualifications Board

(Dansk udgave – release oktober 2011)

Copyright-bestemmelser

Revisionshistorie

Version	Dato	Bemærkninger
ISTQB 2010	31-Okt-2011	Dansk oversættelse (release)
ISTQB 2011	28-Aug-2011	Dansk oversættelse (præ-release).
ISTQB 2010	I kraft 1-Apr-2011	Certified Tester, pensum for Foundation-niveauet Foundation Level Syllabus Vedligeholdelsesudgivelse - se Tillæg E – Releasenotat
ISTQB 2010	I kraft 30-Mar-2010	Certified Tester, pensum for Foundation-niveauet Foundation Level Syllabus Vedligeholdelsesudgivelse - se Tillæg E – Releasenotat
ISTQB 2007	01-Maj-2007	Certified Tester, pensum for Foundation-niveauet Foundation Level Syllabus vedligeholdelsesudgivelse
ISTQB 2005	01-Juli-2005	Certified Tester, pensum for Foundation-niveauet
ASQF V2.2	Juli-2003	ASQF Syllabus Foundation Level Version 2.2 "Lehrplan „Grundlagen des Softwaretestens"
ISEB V2.0	25-Feb-1999	ISEB Software Testing Foundation Syllabus V2.0 25. februar 1999

Indholdsfortegnelse

Bidragsydere

International Software Testing Qualifications Board Working Group Foundation Level (2011-udgave): Thomas Müller (formand), Debra Friedenberg. Hovedteamet takker review teamet (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal) og alle nationale boards for forslag til den nuværende version af pensum.

2010-udgaven er skrevet af International Software Testing Qualifications Board Working Group Foundation Level: Thomas Müller (formand), Rahul Verma, Martin Klonk and Armin Beer. Hovedteamet takker review teamet (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal) og alle nationale boards for deres forslag.

2007-udgaven er skrevet af International Software Testing Qualifications Board Working Group Foundation Level: Thomas Müller (formand), Dorothy Graham, Debra Friedenberg, og Erik van Veenendaal. Hovedteamet takker review teamet (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, og Wonil Kwon) og alle nationale boards for forslag til den nuværende version af pensum.

2005-udgaven er skrevet af International Software Testing Qualifications Board Working Group Foundation Level: Thomas Müller (formand), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson og Erik van Veenendaal. Hovedteamet takker review-teamet og alle nationale boards for deres forslag.

Introduktion til dette pensum

Formålet med dette dokument

Dette pensum danner grundlag for International Software Testing Qualification på Foundation-niveau. International Software Testing Qualifications Board (herefter kaldet ISTQB) stiller pensum til rådighed for de nationale eksaminationsenheder til godkendelse af kursusudbydere og til at udlede eksamens-spørgsmål på deres lokale sprog. Kursusudbydere fremstiller undervisningsmateriale og fastlægger passende undervisningsmetoder til akkreditering på grundlag af pensum, og pensum kan desuden hjælpe kandidater med deres forberedelse til eksamen.

Information om historie og baggrund for pensum findes i Tillæg A.

Certified Tester på Foundation-niveau

Kvalifikationen på Foundation-niveau er beregnet på enhver, der er involveret i test af software. Dette inkluderer personer i roller som f.eks. testere, testanalytikere, testingeniører, testkonsulenter, testansvarlige, bruger-accepttestere og softwareudviklere. Kvalifikationen på Foundation-niveau er også egnet til alle, der ønsker grundlæggende forståelse af softwaretest, som f.eks. projektledere, kvalitetsansvarlige, softwareprogrammører, forretningsanalytikere, it-ledere og virksomhedsledere. Indehavere af Foundation Certificate vil kunne nå et højere kvalifikationsniveau i softwaretest.

Undervisningsmål og vidensniveau

Der gives kognitive niveauer for hvert afsnit i pensum:

- K1: huske
- K2: forstå
- K3: anvende
- K4: analysere

Der er yderligere oplysninger og eksempler om undervisningsmål i Bilag B.

Alle termer, der er oplistet under "Termer" lige efter kapiteloverskrifter, skal kunne huskes (K1), også selvom de ikke er udtrykkeligt nævnt i undervisningsmålet.

Eksamen

Eksamen i Foundation Certificate er baseret på dette pensum. Alle afsnit i pensum er eksamensrelevante. Svar på eksamensspørgsmål kan kræve brug af materiale i mere end et afsnit i pensum.

Eksamensformen er multiple choice-opgaver.

Eksamen kan gennemføres som en del af et godkendt uddannelsesforløb eller tages individuelt (f.eks. i et eksamenscenter eller ved en offentlig eksamen). Gennemførsel af et akkrediteret kursus er ikke en forudsætning for eksamenen.

Akkreditering

Kursusudbydere, hvis kursusmateriale følger dette pensum, kan godkendes af et nationalt nævn, der er godkendt af ISTQB. Retningslinjer for akkreditering kan indhentes fra nævnet eller enheden, der giver akkrediteringen. Et akkrediteret kursus er anerkendt som værende i overensstemmelse med pensum og kan afholde ISTQB-eksamen som en del af kurset.

Yderligere retningslinjer for undervisere findes i Tillæg D.

Detaljeringsgrad

Detaljeringsgraden i pensum gør det muligt at gennemføre ensartet undervisning og eksamination. For at opnå dette formål, består pensum af:

- Generelle undervisningsmål, der beskriver hensigten med Foundation-niveauet
- En liste over oplysninger om undervisningsform med beskrivelse og referencer til ekstra kilder
- Undervisningsmål for hvert vidensområde, der kognitivt beskriver udbytte af undervisning og den tankegang, der skal opnås
- En liste over termer, som de studerende skal kunne huske og forstå
- En beskrivelse af de nøglekoncepter, der skal undervises i, indeholdende kilder som godkendt litteratur eller standarder

Pensum for Foundation-niveauet er ikke en beskrivelse af hele vidensområdet inden for softwaretest. Det afspejler den detaljeringsgrad, der skal dækkes i kurser på Foundation-niveau.

Sådan er pensum organiseret

Der er seks hovedkapitler. Overskriften viser niveauerne for de undervisningsmål, der er dækket i kapitlet. F.eks.:

2. Test igennem hele softwarens livscyklus (K2)

Denne titel viser, at kapitel 2 har undervisningsmål på K1 (antages, når der er vist et højere niveau) og K2 (men ikke K3), og det forventes, at det tager 115 minutter at undervise i kapitlets materiale. I hvert kapitel er der et antal afsnit. Hvert afsnit har også undervisningsmål og den fornødne tid angivet. Underafsnit, der ikke har nogen tidsangivelse, er indeholdt i den tid, der er angivet for det samlede afsnit.

1. Testprincipper (K2)	155 minutter

Undervisningsmål for grundlæggende test
Formålene viser, hvad man skal kunne efter fuldførelse af hvert enkelt modul.

1.1. Hvorfor er test nødvendig? (K2)

LO-1.1.1 Beskrive med eksempler hvordan en defekt kan skade en person, miljøet eller en virksomhed. (K2)
LO-1.1.2 Skelne mellem rodårsagen til en defekt og effekten heraf. (K2)
LO-1.1.3 Give årsager til hvorfor det er nødvendigt at udføre test ved at komme med eksempler. (K2)
LO-1.1.4 Beskrive hvorfor test er en del af kvalitetssikring, og give eksempler på, hvordan test bidrager til en højere kvalitet. (K2)
LO-1.1.5 Forklare og sammenligne termerne fejltagelse, (error), defekt, fejl, afvigelse og de tilsvarende termer fejltagelse (mistake) og bug, med eksempler. (K2)

1.2. Hvad er test? (K2)

LO-1.2.1 Huske de almindelige formål for test. (K1)
LO-1.2.2 Give eksempler på formålet med test i forskellige faser af softwarens livscyklus. (K2)
LO-1.2.3 Forklare forskellen på test og debugging. (K2)

1.3. Syv generelle testprincipper (K2)

LO-1.3.1 Forklar de syv grundlæggende principper ved test. (K2)

1.4. Den grundlæggende testproces (K1)

LO-1.4.1 Huske de fem grundlæggende testaktiviteter fra planlægning til testafslutning og hovedopgaverne for hver enkelt testaktivitet. (K1)

1.5. Psykologi i testen (K2)

LO-1.5.1 Husk de psykologiske faktorer som påvirker testens succes (K1)
LO-1.5.2 Stille testerens og udviklerens tankegang op imod hinanden. (K2)

1.1 Hvorfor er test nødvendig? (K2)	20 minutter

Termer

Bug, defekt, fejl, afvigelse, fejltagelse, kvalitet, risiko.

1.1.1. Kontekst i softwaresystemer (K1)

Softwaresystemer fylder mere og mere i vores liv, fra forretningssystemer (f.eks. bankverdenen) til forbrugerprodukter (f.eks. biler). De fleste mennesker har oplevet software, der ikke fungerede som forventet. Software, der ikke fungerer korrekt, kan føre til mange problemer, herunder tab af penge, tid og virksomhedsomdømme. Den kan også være årsag til alvorlig personskade og død.

1.1.2. Årsager til softwarefejl (K2)

En fejltagelse er en menneskelig handling, der kan medføre en defekt (fejl, bug) i programkoden eller i et dokument. Hvis der opstår en defekt i koden, vil systemet ikke gøre det, det skal (eller ikke skal) gøre, hvilket medfører en afvigelse. Defekter i software, systemer eller dokumenter kan skabe afvigelser, men det er ikke alle defekter, der gør dette.

Defekter opstår, fordi vi ikke er perfekte, og fordi der er tidspres, kompleks kode, infrastrukturel kompleksitet, ændret teknologi og/eller samspil mellem mange systemer.

Afvigelser kan også være forårsaget af miljøbetingelser: For eksempel stråling, magnetisme, elektroniske felter, og forurening kan medføre fejl i firmware eller påvirke softwareafviklingen ved at påvirke hardwaretilstande.

1.1.3. Testrollen ved softwareudvikling, vedligeholdelse og drift (K2)

Omhyggelig systemtest og dokumentation kan hjælpe til med at reducere risikoen for problemer under drift og bidrager til softwaresystemets kvalitet, hvis de fundne defekter rettes, før systemet tages i brug.

Softwaretest kan også være nødvendig for at opfylde kontraktmæssige krav, juridiske krav og industrispecifikke standarder.

1.1.4. Test og kvalitet (K2)

Ved hjælp af test er det muligt at måle softwarekvaliteten, hvad angår fundne defekter både for funktionelle og ikke-funktionelle krav og egenskaber (f.eks. pålidelighed, brugbarhed, effektivitet vedligeholdelsesegnethed og portabilitet). Se kapitel 2 for mere information om ikke-funktionel test. For mere information om softwareegenskaber, se dokumentet "Softwareudvikling – Softwareproduktkvalitet" (ISO 9126).

Test kan skabe tillid til softwarekvaliteten, hvis den finder få eller ingen defekter. En rigtig designet og gennemført test reducerer det overordnede risikoniveau i et system. Når der findes defekter ved testen og de bliver rettet, giver det en bedre softwarekvalitet.

Det er nødvendigt at lære af tidligere projekter. Ved at forstå rodårsagen til fundne defekter i andre projekter, kan man forbedre processerne, og dermed kan man forhindre, at sådanne defekter opstår igen. Dermed forbedrer man kvaliteten i fremtidige systemer. Dette er et aspekt af kvalitetssikring.

Test bør være integreret som en del af kvalitetssikringen (dvs. sideløbende med udvikling af standarder, uddannelse og analyse af defekter).

1.1.5. Hvor meget test er tilstrækkelig? (K2)

Når man skal beslutte, hvor megen test, der er behov for, skal man tage hensyn til risikoniveauet, herunder tekniske, sikkerhedsmæssige og forretningsmæssige risici og projektets begrænsninger, med hensyn til tid og penge. Risiko drøftes yderligere i kapitel 5.

Testen bør give tilstrækkelige information til interessenterne til at de kan træffe beslutninger om frigivelse af softwaren eller systemet på et veldefineret grundlag. Softwaren kan derefter frigives til næste udviklingstrin eller overdrages til kunden.

1.2 Hvad er test? (K2)	30 minutter

Termer

Debugging, krav, review, testcase, testning, testformål.

Baggrund

Det er en almindelig misforståelse, at det at teste kun består af testafvikling, dvs. eksekvering af software. Det er korrekt at testafvikling er en del af det at teste, men testafvikling udgør ikke alle testaktiviteterne.

Der findes testaktiviteter både før og efter testafviklingen. Det inkluderer planlægning og kontrol, valg af testbetingelser, design og udførelse af testcases, kontrol af resultater, evaluering af slutkriterier, rapportering om testprocessen og systemet under testen, og lukning af aktiviteterne efter at en testfase er blevet afsluttet. Test omfatter også review af dokumenter (herunder kildekode) og udførelse af statisk analyse.

Både dynamisk test og statisk test kan opfylde testformålet ved at forbedre det testede system og forbedre udviklings- og testprocessen.

Der kan være forskellige testformål:

- at finde defekter
- at skabe tillid til kvalitetsniveauet og give information
- at forebygge defekter

Den tankeproces, det er at designe tests tidligt i forløbet (verifikation af testgrundlag via testdesign) kan hjælpe med til at forebygge, at defekter indføres i koden. Review af krav og andre dokumenter og identifikationen og løsning af problemer hjælper også med at forebygge, at defekter opstår i koden.

Forskellige typer af test opfylder forskellige formål. F.eks. kan hovedformålet i udviklingstest (komponent-, integrations- og systemtest) være at finde så mange afvigelser som muligt, så softwaredefekterne kan identificeres og rettes. I accepttest kan hovedformålet være at få bekræftet, at systemet virker som forventet, for at sikre at de opfylder kravene. I nogle tilfælde kan hovedformålet med test være at vurdere softwarekvaliteten (uden hensigt om at rette defekter) for at give informationer til interessenter om risikoen ved at frigive systemet på et givet tidspunkt. Vedligeholdelsestest indebærer ofte test af, at der ikke er opstået nye defekter ved udvikling af ændringerne. Ved driftstest kan hovedformålet være at vurdere systemegenskaber som f.eks. pålidelighed og tilgængelighed.

Debugging og test er ikke det samme. Test kan vise afvigelser, der er forårsaget af defekter. Debugging er den udviklingsaktivitet, der finder årsagen til en kendt defekt, reparerer koden og kontrollerer, at defekten er rettet korrekt. Efterfølgende gentest foretaget af tester sikrer, at rettelsen rent faktisk retter afvigelsen. Normalt er ansvaret fordelt sådan, at testere tester, og udviklere debugger.

Testprocessen og dens bestanddele er forklaret i afsnit 1.4.

1.3 Syv generelle testprincipper (K2)	35 minutter

Termer

Udtømmende test.

Principper

Der er igennem de seneste 40 år opstået en række testprincipper, som giver almindelige retningslinjer gældende for al test.

Princip 1 – Test viser tilstedeværelse af defekter

Test kan vise at defekter er til stede, men kan ikke bevise, at der ikke er nogen defekter. Test reducerer sandsynligheden for, at der er endnu ikke afslørede defekter tilbage i softwaren. Det er ikke et bevis for korrekt software, at en test ikke har fundet defekter.

Princip 2 – Udtømmende test er umulig

Udtømmende test, dvs. test af alle kombinationer af inputs og startbetingelser er ikke mulig, bortset fra i uvæsentlige situationer. I stedet for at stræbe efter udtømmende test bør man bruge risikovurdering og prioritering for at fokusere testindsatsen.

Princip 3 – Tidlig test

For at finde fejl tidligt i udviklingsprocessen, skal testaktiviteterne startes så tidligt som muligt i softwarens eller systemudviklingens livscyklus og skal være fokuseret på definerede formål.

Princip 4 – Klynger af defekter

Testindsatsen skal disponeres i forhold til den forventede fejltæthed i modulerne. Når testen er i gang, skal indsatsen disponeres i forhold til den observerede fejltæthed. De fleste af de defekter, der bliver fundet ved test forud for frigivelsen stammer normalt fra få af de testede moduler. Det samme gælder for de defekter, der senere bliver fundet i produktion.

Princip 5 – Pesticid-paradokset

Hvis de samme tests afvikles igen og igen, vil de til sidst ikke finde nye fejl. For at overvinde dette "pesticid-paradoks", skal testcases revideres regelmæssigt, og man skal skrive nye og anderledes tests for at afprøve forskellige dele af softwaren eller systemet for at finde flere defekter.

Princip 6 – Test er afhængig af sammenhængen

Test udføres forskelligt i forskellige sammenhænge. Sikkerhedskritisk software skal f.eks. testes anderledes end et e-handelssite.

Princip 7 – Fravær-af-fejl-fejltagelsen

Det hjælper ikke at finde og rette defekter, hvis det færdige system ikke opfylder brugernes behov og forventninger.

1.4 Grundlæggende testproces (K1)	35 minutter

Termer

Bekræftet test, gentest, udgangskriterier, hændelse, regressionstest, testgrundlag, testbetingelser, testdækning, testdata, testafvikling, testlog, testplan, testprocedure, testpolitik, teststrategi, testsuite, testopsummeringsrapport og test-delprodukter.

Baggrund

Den mest synlige del af testprocessen er afvikling af tests. Men testprocessen omfatter også planlægning, design af testcases, forberedelse af afvikling og evaluering af status. Derfor skal en testplan også omfatte disse aktiviteter.

Den grundlæggende testproces består af følgende hovedaktiviteter:

- testplanlægning og -kontrol
- testanalyse og -design
- testimplementering og -afvikling
- vurdering af slutkriterier og rapportering
- testlukningsaktiviteter

Selv om der er en logisk rækkefølge, kan processerne overlappe hinanden eller finde sted samtidig.

1.4.1 Testplanlægning og kontrol (K1)

Testplanlægning er verifikation af testopdraget, definition af testformålet og specifikation af de nødvendige testaktiviteter, der skal til for at opnå formål og opdrag.

Testkontrol er den løbende aktivitet at sammenligne den aktuelle fremdrift med planen og rapportere status, inklusiv afvigelser fra planen. Testkontrol omfatter de forholdsregler, der er nødvendige for at opfylde projektets opdrag og formål. For at testen kan kontrolleres, må den overvåges gennem hele projektforløbet. Testplanlægning skal tage højde for feedback fra overvågnings- og kontrolaktiviteter.

Testplanlægning og kontrolopgaver er defineret i kapitel 5..

1.4.2 Testanalyse og design (K1)

Testanalyse og design er de aktiviteter, hvor det generelle testformål omdannes til håndgribelige testbetingelser og testcases.
Testanalyse og design har følgende hovedopgaver:

- Review af testgrundlaget (krav, softwarens integritetsniveau[1], arkitektur, design, grænseflader)

[1] Ved softwarens integritetsniveau forstås de karakteristika, som interessenterne forventer at softwaren overholder, f.eks. krav til kompleksitet, risikovurdering, sikkerhedsniveau, ønsket performance, robusthed og pris.

- Evaluering af testbarhed for testgrundlag og testobjekter
- Identificering og prioritering af testbetingelser eller testkrav baseret på analyse af testelementer, specifikation, opførsel og struktur
- Design og prioritering af testcases
- Identifikation af nødvendige testdata, der understøtter testvilkår og testcases
- Design af testmiljø-setup. Identifikation af evt. påkrævet infrastruktur og værktøjer.
- Oprettelse af tovejs-sporbarhed mellem testgrundlag og testcases

1.4.3 Implementering og afvikling af test (K1)

Test-implementering og -afvikling er de aktiviteter, hvor testprocedurer eller scripts er specificeret som en sammensætning af testcases i en planlagt rækkefølge. De omfatter nødvendig information for opsætning af testmiljø og testafvikling.

Test-implementering og -afvikling har følgende hovedopgaver:

- Udvikling, implementering og prioritering af testcases (herunder identifikation af testdata)
- Udvikling og prioritering af testprocedurer, dannelse af testdata og eventuelt forberedelse af testharness og udarbejdelse af automatiserede testscripts
- Sammensætning af sekvenser af test ud fra testprocedurer for effektiv testafvikling
- Verifikation af at testmiljøet er korrekt opsat
- Verifikation og opdatering af tovejs-sporbarhed mellem testgrundlag og testcases
- Afvikling af testprocedurer enten manuelt eller vha. afviklingsværktøjer i den planlagte rækkefølge
- Logning af resultatet af testafviklingen. Registrering af identitet og version af softwaren under test, testværktøjer og test-delprodukter
- Sammenligning af de faktiske resultater med de forventede resultater
- Rapportering af uoverensstemmelser som hændelser og analyse af dem for at fastlægge årsagerne (f.eks. en defekt i koden, i de specificerede testdata, i testdokumentet eller en fejltagelse i udførelsen af testen)
- Gentagelse af testaktiviteter som et resultat af de aktiviteter, der er blevet udført for hver uoverensstemmelse. F.eks. gen-afvikling af en test, der tidligere fejlede for at få bekræftet en rettelse (gentest), afvikling af en rettet test og/eller afvikling af tests for at sikre, at der ikke er opstået defekter i uændrede områder i softwaren, og at en defektrettelse ikke har afdækket andre defekter (regressionstest).

1.4.4 Evaluering af udgangskriterier og rapportering (K1)

Evaluering af udgangskriterier er den aktivitet, hvor testafviklingen vurderes i forhold til de fastlagte formål. Det bør udføres for hvert testniveau.

Evaluering af udgangskriterier har følgende hovedopgaver:

- Kontrol af testlogs i forhold til de udgangskriterier, der er specificeret i testplanlægningen
- Vurdering af, om der er behov for flere tests, eller om de specificerede udgangskriterier bør ændres
- Udarbejdelse af en testopsummeringsrapport til interessenterne

1.4.5 Testlukningsaktiviteter (K1)

Testlukningsaktiviteter indsamler data fra de fuldførte testaktiviteter for at konsolidere erfaring, test-delprodukter, fakta og tal. Testlukningsaktiviteter sker ved projekt milepæle som f.eks når et softwaresystem frigives, et testprojekt er fuldført (eller annulleret), når man er nået til en milepæl, eller når en vedligeholdelsesrelease er fuldført.

Testlukningsaktiviteter indeholder følgende hovedopgaver:

- Kontrol af hvilke af de planlagte leverancer der er leveret
- Lukning af hændelsesrapporter eller oprettelse af ændringsregistreringer for dem der stadig måtte være åbne
- Dokumentation af godkendelsen af systemet
- Færdiggørelse og arkivering af test-delprodukter, testmiljøet og testinfrastrukturen til senere genbrug
- Overdragelse af test-delprodukter til vedligeholdelsesorganisationen
- Analyse af de opnåede erfaringer til brug for fremtidige releases og projekter
- Brug af de opsamlede informationer til forbedring af testmodenhed

1.5 Psykologi i testen (K2)	35 minutter

Termer

Fejlgætning, uafhængighed.

Baggrund

Den tankegang, der skal bruges ved test og review, er forskellig fra den, der skal bruges ved softwareudvikling. Med den rette tankegang er udviklere i stand til at teste deres egen kode. Grunden til at overdrage opgaven til en mere uafhængig tester er for at fokusere indsatsen og få ekstra udbytte i form af f.eks. en uafhængig vurdering fra uddannede testressourcer. Uafhængig test kan udføres på alle testniveauer.

En vis grad af uafhængighed er ofte effektiv til at finde defekter og afvigelser. Uafhængighed er imidlertid ikke en erstatning for kendskab, og udviklere kan finde mange defekter, når de tester deres egen kode. Der kan defineres mange uafhængighedsgrader:

- Tests designet af den eller de personer, der skrev softwaren under test (lav uafhængighedsgrad)
- Tests designes af en anden person/andre personer (f.eks. fra et udviklingsteam)
- Tests designet af en person/personer fra en anden organisationsenhed (f.eks. et uafhængigt testteam) eller testspecialister (f.eks. usability eller performance testspecialister)
- Tests designes af en person/personer fra en anden organisation eller virksomhed (dvs. ved outsourcing eller certificering af en ekstern enhed)

Mennesker og projekter styres af formål. Mennesker har en tendens til at tilpasse deres planer til de formål, der er sat af ledelsen eller interessenterne, f.eks. at finde defekter eller til at bekræfte, at softwaren virker. Det er derfor vigtigt at fastslå et klart formål for testen.

Fund af afvigelser under test kan opfattes som kritik af produktet og af ophavsmanden. Nogle opfatter derfor test som en destruktiv aktivitet. At søge efter afvigelser i et system kræver nysgerrighed, professionel pessimisme, et kritisk øje, opmærksomhed på detaljen, god kommunikation med udviklingskolleger og erfaring som fejlgætning kan baseres på.

Hvis fejl, defekter eller afvigelser kommunikeres på en konstruktiv måde, kan man undgå dårlig stemning mellem testere, analytikere, designere og udviklere. Dette gælder både for review og for test.

Både testeren og testlederen skal have gode samarbejdsevner for at kunne kommunikere faktuelle oplysninger om defekter, fremdrift og risici på en konstruktiv måde. For ophavsmanden til softwaren eller dokumentet kan defektoplysninger hjælpe dem med at forbedre deres færdigheder. De defekter, der findes og rettes under en test, sparer både tid og penge senere i projektforløbet og reducerer risici ved softwaren.

Der kan opstå kommunikationsproblemer, især hvis testere kun ses som nogle, der er budbringere af uønskede nyheder om defekter. Der er imidlertid mange måder, hvorpå man kan forbedre kommunikation og samarbejdet mellem testere og andre:

- Start med samarbejde hellere end med kamp – gør opmærksom på det fælles mål om bedre kvalitet i softwaren
- Hold fokus på facts uden at kritisere den person, der har skabt softwaren, skriv f.eks. objektive og faktuelle hændelsesrapporter og review-observationer
- Forsøg at forstå den anden persons følelser, og hvorfor vedkommende reagerer, som han/hun gør
- Sørg for, at den anden person har forstået, hvad du har sagt - og omvendt

1.6 Etiske regler	10 minutter

Deltagelse i softwaretest giver enkeltpersoner mulighed for at få indblik i fortrolige og interne oplysninger. Etiske regler er nødvendige blandt andet for at sikre, at oplysningerne ikke bruges på en upassende måde. Samtidig med at de etiske regler for teknikere i ACM og IEEE anerkendes, angiver ISTQB følgende etiske regler:

OFFENTLIGHEDEN – Certificerede softwaretestere skal agere i overensstemmelse med offentlighedens interesser.

KLIENT OG ARBEJDSGIVER – Certificerede softwaretestere skal agere på en måde, der bedst muligt tager hensyn til deres klients og arbejdsgivers interesser og er konsistent med offentlighedens interesser.

PRODUKT – Certificerede softwaretestere skal sikre, at de leverancer, de udarbejder (om de produkter og systemer, de tester), opfylder de højest mulige professionelle standarder.

VURDERING – Certificerede softwaretestere skal opretholde integritet og uafhængighed i deres professionelle vurdering.

STYRING – Certificerede softwaretestansvarlige og -ledere skal bidrage med og fremme en etisk tilgang til styring af softwaretest.

ERHVERV– Certificerede softwaretestere skal fremme erhvervets integritet og ry i overensstemmelse med offentlighedens interesse.

KOLLEGER – Certificerede softwaretestere skal være fair og støttende over for deres kolleger og fremme samarbejdet med softwareudviklerne.

PERSONLIGT – Certificerede softwaretestere skal deltage i livslang indlæring med hensyn til udøvelsen af deres erhverv, og de skal fremme en etisk tilgang til udøvelsen af erhvervet.

Referencer

1.1.5	Black, 2001, Kaner, 2002
1.2	Beizer, 1990, Black, 2001, Myers, 1979
1,3	Beizer, 1990, Hetzel, 1988, Myers, 1979
1.4	Hetzel, 1988
1.4.5	Black, 2001, Craig, 2002
1.5	Black, 2001, Hetzel, 1988

2. Test igennem hele softwarens livscyklus (K2)	115 minutter

Undervisningsmål for test igennem hele softwarens livscyklus
Formålene viser, hvad du skal kunne efter fuldførelse af hvert enkelt modul.

2.1 Softwareudviklingsmodeller (K2)

LO-2.1.1 Forklare sammenhængen mellem udvikling, testaktiviteter og arbejdsprodukter i udviklingslivscyklusen, ved at give eksempler baseret på projekt- og produkttyper (K2)
LO-2.1.2 Vide at softwareudviklingsmodeller skal tilpasses projektets kontekst og produktegenskaberne. (K1)
LO-2.1.3 Huske baggrunden for god test som er anvendelig i enhver livscyklusmodel. (K1)

2.2 Testniveauer (K2)

LO-2.2.1 Sammenligne de forskellige testniveauer: hovedformål, typiske testobjekter, typiske testmål (f.eks. funktionel eller strukturel) og relaterede delprodukter, personer der tester, defekt- og afvigelsestyper, der skal findes. (K2)

2.3 Testtyper (K2)

LO-2.3.1 Sammenligne fire softwaretesttyper (funktionel, ikke-funktionel, strukturel og ændrings-relateret) med eksempler. (K2)
LO-2.3.2 Vide at funktionelle og strukturelle tests forekommer på alle testniveauer. (K1)
LO-2.3.3 Finde og beskrive ikke-funktionelle testtyper, baseret på ikke-funktionelle krav. (K2)
LO-2.3.4 Finde og beskrive testtyper, baseret på analysen af softwaresystemets struktur eller arkitektur. (K2)
LO-2.3.5 Beskrive formålet med gentest og regressionstest. (K2)

2.4 Vedligeholdelsestest (K2)

LO-2.4.1 Sammenligne vedligeholdelsestest (test af et eksisterende system) med test af et nyt program, hvad angår testtyper, testudløsere og testomfang. (K2)
LO-2.4.2 Angive årsager til vedligeholdelsestest (ændring, konvertering og lukning). (K1)
LO-2.4.3 Beskrive den rolle regressionstest og effektanalyse har i vedligeholdelse. (K2)

2.1 Softwareudviklingsmodeller (K2)	20 minutter

Termer

Hyldesoftware, iterativ og inkrementel udviklingsmodel, testniveau, validering, verificering, V-model.

Baggrund

Test eksisterer ikke i isolation. Testaktiviteter er knyttet til softwareudviklingsaktiviteter. Forskellig modeller for udvikling kræver forskellige tilgange til test.

2.1.1 V-model (sekventiel udviklingsmodel) (K2)

En almindelig V-modeltype anvender fire testniveauer svarende til de fire udviklingsniveauer:

- Komponenttest (unittest)
- Integrationstest
- Systemtest
- Accepttest

En V-model kan dog have flere, færre eller andre udviklingsniveauer, afhængig af projekt og softwareprodukt. Der kan f.eks. være komponentintegrationstest efter komponenttest, og systemintegrationstest efter systemtest.

Softwarearbejdsprodukter (som f.eks. forretningsscenarier eller usecases, kravspecifikationer, designdokumenter og kode), der er produceret under udvikling, er ofte grundlag for test på et eller flere testniveauer. Referencer vedr. generiske arbejdsprodukter omfatter Capability Maturity Model Integration (CMMI) eller "Softwarelivscyklusprocesser" (IEEE/IEC 12207). Verificering, validering og tidligt testdesign kan udføres under udvikling af softwarearbejdsprodukterne.

2.1.2 Iterativ og inkrementel udviklingsmodel (K2)

Iterativ og inkrementel udvikling er processen med etablering af krav, design, opbygning og test af et system, udført som en serie korte udviklingscyklusser. Eksempler: Prototypefremstilling, Rapid Application Development (RAD), Rational Unified Process (RUP) og agile udviklingsmodeller. Det endelige system, der er fremstilles i en iteration, kan testes på adskillige testniveauer i hver iteration. Et inkrement, der føjes til andre, tidligere udviklede, danner et voksende delsystem, der også bør testes. Regressionstest bliver vigtigere for hver iteration. Verificering og validering kan udføres på hvert inkrement.

2.1.3 Test i en livscyklusmodel (K2)

I enhver livscyklusmodel er der adskillige karakteristika for en god test:

- For hver udviklingsaktivitet er der en tilsvarende testaktivitet
- Hvert testniveau har et testformål, der gælder det specifikke niveau
- Analyse og design af tests til et bestemt testniveau skal starte under den tilsvarende udviklingsaktivitet
- Testere bør involveres i review af dokumenterne, så snart et udkast er tilgængeligt i udviklingslivscyklusen

Testniveauer kan kombineres og organiseres alt efter projektets eller systemarkitekturens natur. Ved f.eks. integration af hyldesoftware i et system, kan køberen udføre integrationstest på systemniveau (f.eks. integration til infrastruktur og andre systemer, eller anden systemanvendelse) og accepttest (funktionel og/eller ikke-funktionel og brugertest og/eller driftstest).

2.2 Testniveauer (K2)	40 minutter

Termer

Alfatest, betatest, komponenttest, drivere, felttest, funktionelle krav, integration, integrationstest, ikke-funktionelle krav, robusthedstest, stubbe, systemtest, testmiljø, testniveau, testdrevet udvikling , brugeraccepttest.

Baggrund

For hvert testniveau må man identificere følgende: Testniveauets generiske formål, delprodukter der refereres for udledning af testcases (dvs. testgrundlag), testobjekt (dvs. hvad der skal testes), typiske defekter og afvigelser, krav til testharness og værktøjsstøtte og specifikke metoder og ansvar.

I testplanlægningen skal man også overveje test af systemkonfigurationsdata.

2.2.1 Komponenttest (K2)

Testgrundlag:

- Komponentkrav
- Detaljeret design
- Kode

Typiske testobjekter:

- Komponenter
- Programmer
- Datakonvertering/migreringsprogrammer
- Databasemoduler

Komponenttest (også kendt som enheds-, modul- eller programtest) søger efter defekter i og kontrollerer funktionen af software (f.eks. moduler, programmer, objekter, klasser etc.), der kan testes individuelt. Afhængigt af konteksten i udviklingslivscyklussen og systemet kan det gøres isoleret fra resten af systemet. Man kan anvende stubbe, drivere og simulatorer.

Komponenttest kan indeholde test af funktionalitet og specifikke ikke-funktionelle egenskaber, f.eks. ressourceopførsel (bl.a. hukommelseslæk) eller robusthedstest såvel som strukturel test (f.eks. forgreningsdækning). Testcases tager udgangspunkt i arbejdsprodukter som f.eks. specifikation af komponenten, softwaredesignet eller datamodellen.

Typisk foregår komponenttest med adgang til den kode, der testes og med støtte fra udviklingsmiljøet, som f.eks. en enhedstest-ramme eller et debugging-værktøj og involverer i praksis ofte den programmør, der skrev koden. Defekter rettes typisk uden formel hændelsesregistrering, så snart de er fundet.

En metode i komponenttest er at forberede og automatisere testcases før kodning. Det kaldes en test-først-metode eller testdrevet udvikling. Denne metode er meget iterativ og er baseret på en

cyklus med udvikling af testcases, derpå fremstilling og integration af små stykker kode, afvikling af komponenttests, derefter at rette enhver defekt og at gentage test og rettelse indtil alle dele af koden består testen.

2.2.2 Integrationstest (K2)

Testgrundlag:

- Software og systemdesign
- Arkitektur
- Arbejdsforløb
- Usecases

Typiske test objekter:

- Delsystemer
- Database-implementering
- Infrastruktur
- Grænseflader
- Systemkonfiguration og konfigurationsdata

Integrationstest tester grænseflader mellem komponenter, tester samspil med andre dele i et system, som f.eks. styresystem, filsystem, hardware og tester grænseflader mellem systemer.

Der kan være mere end ét niveau af integrationstest, og det kan udføres på testobjekter af forskellig størrelse. F.eks.:

1. Komponentintegrationstest tester samspillet mellem softwarekomponenter og udføres efter komponenttesten
2. Systemintegrationstest tester samspillet mellem forskellige systemer og mellem hardware og software og kan udføres efter systemtest. I dette tilfælde kan udviklingsorganisationen måske kun kontrollere den ene side af grænsefladen, hvad der kan vurderes som en risiko. Forretningsprocesser, der er implementeret som arbejdsforløb, kan involvere en række systemer. Der kan være store problemer på tværs af platforme.

Jo større omfanget af integrationsopgaven er, jo sværere bliver det at isolere defekter til en specifik komponent eller et system, hvilket kan føre til øget risiko.

Systematiske integrationsstrategier kan være baseret på systemarkitekturen (som f.eks. top-down og bottom-up), funktionelle opgaver, rækkefølgen af transaktioner og andre system- og komponentaspekter. For at gøre det lettere at isolere fejl og finde defekter tidligt, bør integration normalt være inkrementel fremfor "big-bang".

Test af specifikke, ikke-funktionelle egenskaber (f.eks. performance) kan være indeholdt i integrationstest.

I integrationstest koncentrerer testerne sig udelukkende om selve integrationen. Hvis de f.eks. integrerer et modul A med et modul B, er de interesseret i at teste kommunikationen mellem modulerne, ikke funktionaliteten af de enkelte moduler. Der kan anvendes både funktionelle og strukturelle metoder.

Ideelt set bør testere forstå arkitekturen og have indflydelse på integrationsplanlægningen. Hvis integrationstests planlægges før komponenter eller systemer er bygget, kan disse bygges i den rækkefølge, der vil give den mest effektive test.

2.2.3 Systemtest (K2)

Testgrundlag:

- System- og software-kravspecifikation
- Usecases
- Funktionel specifikation
- Risikoanalyse-rapporter

Typiske testobjekter:

- System-, bruger- og betjeningsmanualer
- Systemkonfiguration og konfigurationsdata

Systemtest drejer sig om den adfærd for systemet eller produktet, der er defineret af omfanget af et udviklingsprojekt eller program.

Ved systemtest bør testmiljøet så vidt muligt svare til det endelige mål- eller produktionsmiljø for bedre at kunne finde miljøspecifikke afvigelser.

Systemtest kan indeholde tests, der er baseret på risici og/eller kravspecifikationer, forretnings-processer, usecases eller andre beskrivelser på højniveau af systemadfærd, samspil med styresystem og systemressourcer.

Systemtest bør undersøge både funktionelle og ikke-funktionelle krav til systemet, samt datakvaliteten. Krav kan både bestå af tekst og modeller. Testere skal også håndtere ufuldstændige eller udokumenterede krav. Systemtest af funktionelle krav starter med brug af de bedst egnede specifikationsbaserede teknikker (Black-box) for det aspekt af systemet, der skal testes. Der kan f.eks. dannes en beslutningstabel for kombinationer af de effekter, der er beskrevet i forretningsreglerne. Strukturbaserede teknikker (white-box) kan derefter anvendes til at vurdere grundigheden af testen vedrørende et strukturelement, som f.eks. menu-struktur eller webside-navigation. (Se kapitel 4.)

Systemtest udføres ofte af et uafhængigt testteam.

2.2.4 Accepttest (K2)

Testgrundlag:

- Brugerkrav
- Systemkrav
- Usecases
- Forretningsprocesser
- Risikoanalyserapporter

Typiske testobjekter:

- Forretningsprocesser for det fuldt integrerede system

- Driftstests og vedligeholdelsesprocesser
- Brugerprocedures
- Formular
- Rapporter
- Konfigurationsdata

Accepttest er ofte kundernes eller systembrugernes ansvar. Andre interessenter kan også være involveret.

Målet med accepttest er at skabe tillid til systemet, dele af systemet eller specifikke ikke-funktionelle egenskaber i systemet. Det er ikke hovedformålet at finde defekter. Accepttest kan vise systemets parathed til anvendelse og brug, skønt det ikke nødvendigvis er det sidste testniveau. Der kan f.eks. komme en systemintegrationstest i stor skala efter accepttesten.

Accepttest kan også forekomme som mere end blot et enkelt testniveau, f.eks.:

- Man kan accepteste hyldesoftware,når det installeres eller integreres
- Accepttest af en komponents brugervenlighed kan være udført under komponenttest
- Accepttest af en ny funktionel forbedring kan komme før systemtest

Typiske former for accepttest indeholder følgende:

Brugeraccepttest

Verificerer typisk systemets egnethed for virksomhedens brugere.

Driftstest eller driftsaccepttest

Systemadministratorernes accept af systemet, herunder:

- test af backup/gendannelse
- gendannelse ved katastrofe
- brugerhåndtering
- vedligeholdelsesopgaver
- dataload og migreringsopgaver
- periodisk kontrol af sikkerhedssårbarhed

Kontrakt- og regelaccepttest

Kontraktaccepttest udføres i forhold til en kontrakts acceptkriterier for fremstilling af kundetilpasset software. Acceptkriterierne bør være defineret, når aftalen indgås. Regelaccepttest udføres i forhold til de lovbestemmelser, der skal overholdes, som f.eks. offentlige, juridiske eller sikkerhedsmæssige bestemmelser.

Alfa- og beta-test

Udviklere af hyldesoftware ønsker ofte at få feedback fra mulige eller eksisterende kunder på markedet, før et softwareprodukt frigives for kommercielt salg. Alfatest udføres hos udviklingsorganisationen, men ikke af udviklingsholdet. Betatest udføres af kunder eller potentielle kunder på deres egne lokationer. Betatest kaldes også felttest.

Organisationer kan også bruge andre termer, som f.eks. fabriksaccepttest og accepttest på installationsstedet af systemer, der tidligere er testet og derefter flyttes til et en kundes lokation.

2.3 Testtyper (K2)	40 minutter

Termer

Black-box-test, kodedækning, funktionel test, tværoperationalitetstest, loadtest, vedligeholdelsesegnethedstest, performancetest, flytbarhedstest, pålidelighedstest, sikkerhedstest, specifikationsbaseret test, stresstest, strukturel test, brugervenlighedstest , white-box test.

Baggrund

En gruppe testaktiviteter kan have det formål at verificere softwaresystemet (eller en del af et system), baseret på en specifik grund eller testmål.

En testtype er fokuseret på et bestemt testformål:

- Det kunne være test af en funktion, der udføres af softwaren
- En ikke-funktionel kvalitetsegenskab, som f.eks. pålidelighed eller brugervenlighed
- Strukturen eller arkitekturen af softwaren eller systemet, eller
- Relateret til ændringer, dvs. bekræftelse af at defekter er rettet (gentest) og søgning efter utilsigtede ændringer (regressionstest).

En model af softwaren kan udvikles og/eller anvendes til strukturel test (f.eks. en kontrolflowmodel eller menustrukturmodel), til ikke-funktionel test (f.eks. performancemodel, brugervenlighedsmodel eller modellering af sikkerhedstrusler), til funktionel test (f.eks. procesflowmodel, en tilstandsovergangsmodel eller til en specifikation i klart sprog).

2.3.1 Test af funktion (funktionstest) (K2)

De funktioner, som et system, undersystem eller en komponent skal udføre, kan være beskrevet i arbejdsprodukter, som f.eks. kravspecifikation, usecases, en funktionel specifikation, eller de kan være udokumenterede. Funktionerne er "hvad" systemet gør.

Funktionelle tests er baseret på funktioner og egenskaber (beskrevet i dokumenter eller forstået af testerne), og deres interoperalitet med specifikke systemer, og de kan udføres på alle testniveauer. For eksembel kan tests til komponenter være baseret på en komponentspecifikation.

Specifikationsbaserede teknikker kan bruges til at udlede testbetingelser og testcases fra funktionaliteten af softwaren eller systemet (se kapitel 4). Funktionel test tager softwarens eksterne adfærd i betragtning (black-box-test).

En type af funktionel test, sikkerhedstest, undersøgelser de funktioner (f.eks. firewall), der er relateret til opdagelse af trusler, som f.eks. vira, fra ondsindede personer udefra. En anden type funktionel test, interoperationel test, evaluerer softwarens evne til at interagere med en eller flere udvalgte komponenter eller systemer.

2.3.2 Test af ikke-funktionelle softwareegenskaber (ikke-funktionel test) (K2)

Ikke-funktionel test omfatter, men er ikke begrænset til, performancetest, loadtest, stresstest, brugervenlighedstest, vedligeholdelsesegnethedstest, pålidelighedstest og flytbarhedstest. Det er testen, der viser, "hvordan" systemet fungerer.

Ikke-funktionel test kan udføres på alle testniveauer. Ordet ikke-funktionel test beskriver de tests, der er nødvendige for at måle de system- og softwareegenskaberne, der kan opgøres mængdemæssigt på en variabel skala, som f.eks. svartid for performancetest. Disse tests kan henvise til en kvalitetsmodel, som f.eks. den der er defineret i "Software Engineering – Software Product Quality" (ISO 9126). Ikke-funktionel test bedømmer softwarens reaktion set udefra, og i de fleste tilfælde anvendes black-box testdesignteknik.

2.3.3 Test af softwarestruktur og -arkitektur (strukturel test) (K2)

Strukturel test (white-box test) kan udføres på alle testniveauer. Strukturelle teknikker anvendes bedst efter specifikationsbaserede teknikker. De kan måle testens grundighed ved at bestemme dækningen for en strukturtype.

Dækning er det omfang en struktur er blevet berørt af en sekvens af testcases, udtrykt som den procentdel af elementerne, der er dækket. Hvis dækningen ikke er 100%, kan man designe flere tests og dermed øge dækningen. Dækningsteknikker er beskrevet i kapitel 4.

Man kan anvende værktøjer til måling af dækning af kodeelementer på alle testniveauer, især ved komponenttest og komponentintegrationstest. Kodeelementerne kan f.eks. være instruktioner eller beslutninger. Strukturel test kan baseres på systemets arkitektur, som f.eks. et kalde-hierarki.

Strukturelle testmetoder kan også anvendes ved system-, systemintegrations- eller accepttest-niveauer (f.eks. til forretningsmodeller eller menustrukturer).

2.3.4 Test i forbindelse med ændringer: gentest og regressionstest (K2)

Efter en defekt er registreret og rettet, bør softwaren testes igen for at få bekræftet, at den oprindelige defekt er fjernet korrekt. Dette kaldes gentest eller bekræftelsestest. Debugging (lokalisering og rettelse af en defekt) er en udviklingsaktivitet, ikke en testaktivitet.

Regressionstest er en gentagelse af testen for et allerede testet program, efter der er foretaget en ændring. Regressionstesten skal afsløre, om der er opstået eller afdækket nye defekter som følge af ændringerne. Disse defekter kan være i softwaren, der testes, eller i en anden, relateret eller ikke-relateret softwarekomponent. Regressionstesten udføres, når softwaren eller miljøet ændres. Omfanget af en regressionstest er baseret på risikoen, der er ved ikke at finde defekter i software, der har fungeret hidtil.

Tests bør være repeterbare, hvis de skal anvendes til gentest og til støtte for regressionstest.

Regressionstest kan udføres på alle testniveauer og omfatter funktionel, ikke-funktionel og strukturel test. Der kan være behov for at afvikle sekvenser af regressionstest mange gange, og der er normalt kun brug for få ændringer. Derfor er regressionstest et oplagt emne for automatisering.

2.4 Vedligeholdelsestest (K2)	15 minutter

Termer

Effektanalyse, vedligeholdelsestest .

Baggrund

Når et softwaresystem er taget i brug, anvendes det ofte i mange år eller årtier. I denne periode bliver systemet, dets konfigurationsdata, eller dets miljø ofte rettet, ændret eller udvidet. Rettidig planlægning af releases er afgørende for en vellykket vedligeholdelsestest. Man må skelne mellem planlagte frigivelser og hot fixes. Vedligeholdelsestest udføres på et eksisterende styresystem ved ændring, flytning eller lukning af software eller system.

Ændringer omfatter planlagte forbedringsændringer (f.eks. release-baserede), rettelses- og nød-ændringer, og ændringer i omgivelserne, som f.eks. planlagt opgradering af styresystem eller database, planlagt opgradering af hyldesoftware, eller patches for sårbarheder i styresystemet.

Vedligeholdelsestest for flytning (f.eks. fra en platform til en anden) bør omfatte både driftstest af det nye miljø og af den ændrede software. Flytningstest (konverteringstest) er nødvendig, når data fra andre systemer skal flyttes ind i det system som vedligeholdes.

Vedligeholdelsestest ved lukning af et system kan indeholde test af dataflytning eller arkivering, hvis der er tale om en lang dataopbevaringsperiode.

Ud over at teste, hvad der er ændret, så indeholder vedligeholdelsestest også regressionstest af de dele af systemet, der ikke er ændret. Omfanget af vedligeholdelsestest afhænger af risikoen ved ændringen, størrelsen af det eksisterende system og størrelsen af ændringen. Afhængigt af behovet kan en vedligeholdelsestest udføres på alle testniveauer for alle testtyper.

Bestemmelsen af hvordan det eksisterende system kan blive påvirket af ændringer kaldes en effektanalyse, og den anvendes til at bestemme omfanget af regressionstesten.

Vedligeholdelsestest kan være vanskelig, hvis specifikationerne ikke er opdaterede, eller hvis der mangler testere med domæneviden.

Referencer

2.1.3 CMMI, Craig, 2002, Hetzel, 1988, IEEE 12207
2.2 Hetzel, 1988
2.2.4 Copeland, 2004, Myers, 1979
2.3.1 Beizer, 1990, Black, 2001, Copeland, 2004
2.3.2 Black, 2001, ISO 9126
2.3.3 Beizer, 1990, Copeland, 2004, Hetzel, 1988
2.3.4 Hetzel, 1988, IEEE 829
2.4 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE 829

3. Statiske teknikker (K2)	60 minutter

Undervisningsmål for statiske teknikker
Formålene viser, hvad du skal kunne efter fuldførelse af hvert enkelt modul.

3.1 Statiske teknikker og testproces (K2)

LO-3.1.1 Kende softwarearbejdsprodukter, der kan undersøges ved hjælp af forskellige statiske teknikker. (K1)
LO-3.1.2 Beskrive vigtigheden og værdien af at overveje statiske teknikker til vurdering af softwarearbejdsprodukter. (K2)
LO-3.1.3 Forklare forskellen mellem statiske og dynamiske teknikker, overveje formål, typer af defekter der kan identificeres, og den rolle disse teknikker har i software livscyklussen. (K2)

3.2 Reviewproces (K2)

LO-3.2.1 Huske faser, roller og ansvar for et typisk formelt review. (K1)
LO-3.2.2 Forklare forskellen mellem forskellige reviewtyper: uformelt review, teknisk review, walkthrough og inspektion. (K2)
LO-3.2.3 Forklare faktorer for en vellykket gennemførsel af et review. (K2)

3.3 Statisk analyse med værktøjer (K2)

LO-3.3.1 Huske typiske defekter og fejl, der findes ved statisk analyse, og sammenligne dem med reviews og dynamisk test. (K1)
LO-3.3.2 Beskrive, med eksempler, typiske fordele ved statisk analyse. (K2)
LO-3.3.3 Opliste typiske kode- og designdefekter, der kan findes med statisk analyseværktøjer. (K1)

3.1 Statiske teknikker og testproces (K2)	15 minutter

Termer

Dynamisk test, statisk test.

Baggrund

Til forskel fra dynamisk test, som kræver afvikling af software, består statisk test af en manuel gennemgang (et review) eller en automatiseret analyse (statisk analyse) af koden eller anden projektdokumentation uden afvikling af koden.

Reviews er en måde at teste softwarearbejdsprodukter (herunder kode) og kan udføres langt tidligere end afviklingen af dynamisk test. Defekter, der findes under reviews tidligt i livscyklus, er ofte meget billigere at rette end dem, der findes ved afvikling af tests. Et review kan f.eks. afsløre defekter i krav.

Et review kan udføres som en helt manuel aktivitet, men der findes også værktøjer til dette. Den vigtigste manuelle aktivitet er at undersøge et arbejdsprodukt og lave kommentarer til det. Alle softwarearbejdsprodukter kan gennemgås, herunder kravspecifikationer, designspecifikationer, kode, testplaner, testspecifikationer, testcases, testscripts, brugervejledninger og websider.

Fordele ved reviews er tidlig fejlfinding og –rettelse, forbedringer i udviklingseffektivitet, reduceret udviklingstid, reducerede testudgifter og –tid, reduktion i levetidsomkostninger, færre defekter og forbedret kommunikation. Reviews kan finde mangler, som en dynamisk test sandsynligvis ikke finder, f.eks. mangler i krav.

Reviews, statisk analyse og dynamisk test har samme formål – at finde defekter. De er komplementerende: de forskellige teknikker kan på en effektiv måde finde forskellige defekttyper. Sammenlignet med dynamisk test finder statisk test snarere grundene til fejl (defekter) end selve fejlen.

Typiske defekter, der er lettere at finde ved reviews end ved dynamisk test er: Afvigelser fra standarder, kravdefekter, designdefekter, utilstrækkelig vedligeholdelsesegnethed og ukorrekte grænseflade-specifikationer.

3.2 Reviewproces (K2)	25 minutter

Termer

Indgangskriterier, formelt review, uformelt review, inspektion, metrikker, moderator/ inspektionsleder, kollegareview , reviewer, referent, teknisk review, walkthrough.

Baggrund

De forskellige typer af reviews varierer fra uformelle, uden skriftlig vejledning til reviewerne, til meget formelle, kendetegnet ved at inkludere reviewteam, dokumenterede resultater af review, og dokumenterede procedurer for reviewgennemførelse. Graden af formalisering i en reviewproces afhænger af f. eks. modenhed i udviklingsprocessen, juridiske eller lovregulerede krav, og behovet for et revisionsspor.

Måden, som et review udføres på, afhænger af formålet, f.eks. at finde defekter, at opnå forståelse, at skabe diskussion, eller at opnå konsensus om en beslutning.

3.2.1 Faser ved et formelt review (K1)

Et typisk formelt review har følgende hovedfaser:

1. Planlægning
   - Fastlæggelse af reviewkriterier
   - Valg af deltagere
   - Tildeling af roller
   - Definition af indgangs- og udgangskriterier for mere formelle reviewtyper (f.eks. inspektion)
   - Valg af hvilke dele af dokumenter, der skal reviewes.
   - Kontrol af indgangskriterier (for mere formelle reviewtyper)
2. Kick-off:
   - Fordeling af dokumenter
   - Forklaring af formål, proces og dokumenter for deltagerne
3. Individuel forberedelse
   - Forberedelse før reviewmødet, ved at review dokumentet
   - Notering af mulige defekter, spørgsmål og bemærkninger.
4. Granskning/evaluering/registrering af resultater (reviewmøde)
   - Diskussion eller logning med dokumenterede resultater eller referater (for mere formelle reviewtyper).
   - Mødedeltagerne kan notere defekter, lave anbefalinger til håndtering af defekter eller træffe beslutninger om defekterne.
   - Granskning/evaluering og registrering af problemer under fysiske møder eller sporing af gruppens elektroniske kommunikation
5. Gen-bearbejdning
   - Rettelse af fundne defekter (udføres typisk af ophavsmanden)
   - Registrering af opdaterede status for defekterne (i formelle reviews)
6. Opfølgning
   - Kontrol af at der er gjort noget ved defekterne
   - Indsamling af metrikker

- Kontrol af udgangskriterier (ved mere formelle reviewtyper).

3.2.2 Roller og ansvar (K1)

Et typisk formelt review indeholder følgende roller:

- Manager: bestemmer udførelsen af reviews, tildeler tid i projektplaner og tager stilling til om målet med reviewet er nået
- Moderator: leder reviewet af dokumenterne, herunder planlægning, afvikling af mødet og opfølgning efter mødet. Om nødvendigt kan moderatoren mægle mellem modstående synspunkter. Moderatoren er ofte den person, der bidrager mest til at et review lykkes.
- Ophavsmand: forfatteren eller den hovedansvarlige person for de dokumenter, der skal reviewes
- Reviewere: personer med en specifik teknisk eller forretningsmæssig baggrund (også kaldet checkere eller inspektorer) der, efter den nødvendige forberedelse, finder og beskriver observationer (f.eks. defekter) i det undersøgte produkt. Reviewere bør vælges, så de repræsenterer forskellige perspektiver og roller i processen, og de bør deltage i eventuelle reviewmøder.
- Referent (eller recorder): dokumenterer alle observationer, problemer og åbne punkter, der bliver berørt under mødet

Ved at se på softwareprodukterne og de relaterede arbejdsprodukter fra forskellige perspektiver og anvende tjeklister, kan reviews gøres mere effektive. Tjeklister kan hjælpe med at afsløre problemer, f. eks. baseret på perspektiver som bruger, vedligeholder, tester eller driftsansvarlig, eller en tjekliste med typiske problemer ved krav.

3.2.3 Reviewtyper (K2)

Et enkelt dokument kan være genstand for mere end ét review. Hvis det er tilfældet, kan rækkefølgen variere. Der kan f.eks. udføres et uformelt review før et teknisk review, eller der kan udføres en inspektion af en kravspecifikation før en walkthrough med deltagelse af virksomhedens kunder. Hovedegenskaber, muligheder og formål for almindelige reviewtyper er:

Uformelt review

- ingen formel proces
- der kan være parprogrammering eller en teknisk leder, der reviewer design og kode
- reviewet kan eventuelt dokumenteres
- kan variere i brugbarhed afhængig af revieweren
- hovedformål: en billig måde at få et udbytte

Walkthrough

- møde ledet af ophavsmand
- scenarier, dry runs, kollegagruppe
- møder uden fastlagt varighed
  - eventuelt et formøde til forberedelse af reviewere
  - eventuelt forberedelse af review-rapport med en liste over beservationer.
- eventuelt en referent (som ikke er ophavsmand)
- kan i praksis variere fra meget uformelt til meget formelt

- hovedformål: læring, fælles forståelse, at finde defekter

Teknisk review

- dokumenteret, defineret defekt-søgnings-proces, der inddrager kolleger og tekniske eksperter
- kan udføres som kollegareview uden deltagelse fra ledelsen
- ledes ideelt set af en uddannet moderator (ikke ledet af ophavsmanden)
- forberedelse ved formøde
- eventuelt brug af tjeklister
- forberedelse af en reviewrapport, der indeholder en liste over observationer, beslutningen hvorvidt softwareproduktet opfylder dets krav, og i givet fald, anbefalinger i relation til observationer
- kan i praksis variere fra meget uformelt til meget formelt
- hovedformål: at diskutere, træffe beslutninger, evaluere alternativer, finde defekter, løse tekniske problemer og at kontrollere overensstemmelse med specifikationer og standarder

Inspektion

- ledes af en uddannet moderator (som ikke er ophavsmanden)
- sædvanligvis gennemgang udført af kolleger
- definerede roller
- omfatter opsamling af metrikker
- formel proces baseret på regler og tjeklister
- specificerede indgangs- og udgangskriterier for accept af softwareproduktet
- forberedelse ved formøde
- inspektionsrapport med liste over observationer
- formel opfølgningsproces (eventuel registrering af procesforbedringer)
- eventuelt læser af materialet
- hovedformål: at finde defekter

Walkthroughs, tekniske reviews og inspektioner kan udføres af en gruppe kollegaer på samme organisatoriske niveau. Denne type review kaldes et "kollegareview".

3.2.4 Succesfaktorer for reviews (K2)

Succesfaktorer for reviews omfatter:

- Hvert review har et klart forud defineret formål
- Valg af de rigtige personer i forhold til reviewformålet
- Testere er værdifulde reviewere som bidrager til reviewet og samtidig lærer om produktet. Det gør det muligt for dem at forberede testen på et tidligt tidspunkt.
- De fundne defekter er formuleret objektivt og bliver vel modtaget
- Personlige emner og psykologiske aspekter håndteres så det bliver en positiv oplevelse for ophavsmanden
- Reviewet gennemføres i en atmosfære af tillid, hvor resultatet ikke bliver brugt som en evaluering af deltagerne
- De anvendte reviewteknikker er velegnede til at opnå målet og passer med type og niveau for softwarearbejdsprodukter og reviewere

- Tjeklister eller roller anvendes, hvis de kan medvirke til at øge effektiviteten ved identifikation af defekter
- Der gives undervisning i reviewteknik, især de mere formelle teknikker, som inspektion
- Ledelsen støtter en god reviewproces, f.eks. ved at sørge for passende tid til reviews i projektplanerne
- Der er lagt vægt på læring og procesforbedring

3.3 Statisk analyse vha. værktøjer (K2)	20 minutter

Termer

Oversætter, kompleksitet, kontrolflow, dataflow, statisk analyse.

Baggrund

Formålet ved statisk analyse er at finde defekter i softwarens kildekode og softwaremodellerne. Statisk analyse udføres uden egentlig eksekvering af softwaren. Statisk analyse kan lokalisere defekter, der er svære at finde ved test. Som ved reviews gælder det også for statiske analyser, at de finder defekter snarere end afvigelser. Statiske værktøjer analyserer programkode (f.eks. kontrolflow og dataflow), såvel som genereret output i f.eks. HTML og XML.

Værdien ved statisk analyse er:

- Tidlig opdagelse af defekter forud for afviklingen af den dynamiske test
- Tidlig advarsel om mistænkelige aspekter i kode eller design, som f.eks. et højt kompleksitetstal, vha. metrikker
- Identifikation af defekter, der er svære at finde ved dynamisk test
- Afsløring af afhængigheder og inkonsistenser i softwaremodeller som f.eks. links
- Lettere vedligeholdelse af kode og design
- Forebyggelse af defekter, hvis organisationen tager ved lære af erfaringerne

Typiske defekter fundet ved statiske analyseværktøjer omfatter:

- reference til variable med udefinerede værdier
- inkonsistente grænseflader mellem moduler og komponenter
- Afsløring af variable, der aldrig anvendes, eller som er ukorrekt defineret
- utilgængelig (død) kode
- manglende eller fejlagtig logik (f.eks. uendelige løkker)
- for komplicerede konstruktioner
- overtrædelse af programmeringsstandarder
- sikkerhedssårbarheder
- syntaksovertrædelser i kode og softwaremodeller

Statiske analyseværktøjer anvendes typisk af udviklere (der kontrollerer i forhold til fastlagte regler og programmeringsstandarder) før og under komponent- og integrationstest, eller når koden tjekkes ind i et værktøj til konfigurationsstyring. De benyttes desuden af designere under softwaremodel-arbejdet. Statiske analyseværktøjer kan give et stort antal advarselsmeddelelser, dvs. de skal styres omhyggeligt for at give udbytte.

Oversættere (compilere) tilbyder nogen støtte til statisk analyse, f.eks. beregning af metrikker.

Referencer

3.2 IEEE 1028
3.2.2 Gilb, 1993, van Veenendaal, 2004
3.2.4 Gilb, 1993, IEEE 1028

3.3 van Veenendaal, 2004

4. Testdesignteknik (K4)	285 minutter

Undervisningsmål for testdesignteknik
Formålene viser, hvad du skal kunne efter fuldførelse af hvert enkelt modul.

4.1 Testudviklingsprocessen (K3)

LO-4.1.1 Skelne mellem en testdesign-specifikation, testcase-specifikation og en testprocedure- specifikation (K2)
LO-4.1.2 Sammenligne termerne testbetingelse, testcase og testprocedure (K2)
LO-4.1.3 Bedømme kvaliteten af testcases med hensyn til
- tydelig sporbarhed til kravene
- angivelse af forventet resultat (K2)

LO-4.1.4 Oversætte testcases til en velstruktureret testprocedurespecifikation på et detaljeniveau, der er relevant i forhold til testernes viden (K3)

4.2 Kategorier af testdesignteknik (K2)

LO-4.2.1 Huske årsager til at både specifikationsbaserede (black-box) og strukturbaserede (white-box) metoder til testcasedesign er nyttige, og opliste de almindelige teknikker for hver (K1)
LO-4.2.2 Forklare egenskaber og forskelle mellem specifikationsbaseret test, strukturbaseret test og erfaringsbaseret test (K2)

4.3 Specifikationsbaserede eller black-box-teknikker (K3)

LO-4.3.1 Skrive testcases fra givne softwaremodeller vha. følgende testdesignteknikker: (K3)
- ækvivalenspartitionering
- grænseværdianalyse
- beslutningstabeltest
- tilstandsovergangstest

LO-4.3.2 Forklare hovedformålet med hver af de fire testteknikker, hvilket testniveau og -type, der kunne bruge teknikken, og hvordan dækning vil kunne måles (K2)
LO-4.3.3 Forklare konceptet af usecase test og fordelene herved (K2)

4.4 Strukturbaseret eller white-box-teknikker (K4)

LO-4.4.1 Beskrive konceptet og vigtigheden af kodedækning (K2)
LO-4.4.2 Forklare koncepter for instruktions- og beslutningsdækning og forstå, at disse koncepter også kan anvendes på andre testniveauer end komponenttest (f.eks. for forretningsprocedurer på systemniveau)(K2)
LO-4.4.3 Skrive testcases fra givne kontrolflows vha. følgende testdesignteknikker: (K3)
- instruktionstest
- beslutningstest

LO-4.4.4 Vurdere fuldstændigheden af instruktions- og beslutningsdækning under hensyntagen til de definerede slutkriterier (K4)

4.5 Erfaringsbaserede teknikker (K2)

LO-4.5.1 Huske årsager til skrivning af testcases baseret på intuition, erfaring og kendskab til almindelige defekter (K1)

LO-4.5.2 Sammenligne erfaringsbaserede teknikker med specifikationsbaserede testteknikker (K2)

4.6 Valg af testteknik (K2)

LO-4.6.1 Klassificere testdesignteknikker ud fra hvordan de passer ind i en given kontekst, til testgrundlaget, henholdsvis modeller og softwareegenskaber (K2)

4.1 Testudviklingsprocessen (K3)	15 minutter

Termer

Testcase-specifikation, testdesign, testafviklingsrækkefølge, testprocedure-specifikation, testscript, sporbarhed.

Baggrund

Processen, der er beskrevet i dette afsnit, kan udføres på flere måder, fra meget uformel med lidt eller ingen dokumentation til meget formel (som beskrevet herunder). Graden af formalisering afhænger af konteksten for testen, herunder modenheden af test- og udviklingsprocesser, tidsbegrænsninger, sikkerheds- og myndighedskrav, samt de involverede personer.

Ved testanalyse analyseres den grundlæggende testdokumentation for at afgøre, hvad der skal testes, dvs. identifikation af testbetingelserne. En testbetingelse er et element eller en hændelse, der kan verificeres af en eller flere testcases (dvs. en funktion, transaktion, kvalitetsegenskab eller et strukturelt element).

Ved at etablere sporbarhed fra testbetingelser til specifikationer og krav er det muligt at se virkningen af ændringer i kravene og at se om et sæt af tests dækker kravene. I testanalyse-fasen anvender man den detaljerede testtilgang for at kunne vælge det bedst egnede testdesign, bl.a. baseret på de identificerede risici (se kapitel 5 for flere oplysninger om risikoanalyse).

Testdesignet er en proces hvor testerne beskriver testcases og testdata. En testcase består af et sæt inputværdier, afviklings-startbetingelser, forventede resultater og afviklings-post-betingelser, defineret til at dække en eller flere bestemte testbetingelser. "Standard for Software Test Dokumentation" (IEEE 829-1998) beskriver indholdet af testdesign-specifikationer og testcase-specifikationer.

En testcase skal også indeholde det forventede resultat med output, ændringer i data og tilstande og andre konsekvenser af testen. Hvis de forventede resultater ikke er defineret, så kan et sandsynligt, men fejlagtigt resultat måske fortolkes som om det var korrekt. Derfor bør de forventede resultater defineres forud for testafviklingen.

I løbet af testimplementering er en række testcases udviklet, implementeret, prioriteret og organiseret i testprocedure-specifikationen. Testproceduren (eller et manuelt testscript) specificerer rækkefølgen af handlinger for afviklingen af en test. Hvis tests køres vha. et testafviklingsværktøj, skal rækkefølgen af handlinger specificeres i et testscript (en automatiseret testprocedure).

De forskellige testprocedurer og automatiserede testscripts bliver efterfølgende omdannet til en testafviklingsplan, der definerer rækkefølgen, som de forskellige testprocedurer og testscripts afvikles i, og definerer hvem der skal udføre dem. Testafviklingsplanen medtager faktorer som regressionstests, prioritering og tekniske og logiske afhængigheder.

4.2 Kategorier af testdesignteknikker (K2)	15 minutter

Termer

Black-box testdesignteknik, erfaringsbaserede testdesignteknik, specifikationsbaserede testdesignteknik, strukturbaserede testdesignteknik, white-box testdesignteknik.

Baggrund

Formålet med en testdesignteknik er at identificere testbetingelser og testcases.

Det er en klassisk skelnen at angive testteknikker som black-box eller white-box. Black-box teknik (også kaldet specifikationsbaserede teknik) er en metode til at udlede og vælge testbetingelser, testcases og testdata baseret på en analyse af dokumentationen for testgrundlaget. Den omfatter både funktionel og ikke-funktionel test. Black-box-test anvender per definition ingen information om den interne struktur i det system eller den komponent, der skal testes. White-box-teknik (også kaldet strukturel eller strukturbaseret teknik) er baseret på en analyse af den interne struktur af komponenten eller systemet. Både black-box og white-box-test kan trække på erfaringer hos udviklere, testere og brugere til at afgøre hvad der skal testes.

Nogle teknikker falder helt klart i en enkelt kategori, andre har elementer af flere kategorier.

Dette pensum refererer til specifikationsbaserede designmetoder som black-box-teknik og strukturbaserede metoder som white-box-teknik. Derudover behandles erfaringsbaseret testteknik.

Almindelige egenskaber for black-box-teknik omfatter:

- Modeller, enten formelle eller uformelle, anvendes til specifikation af det problem, der skal løses af softwaren eller af komponenterne
- Fra disse modeller kan man systematisk udlede testcases

Almindelige egenskaber for white-box-teknik omfatter:

- Oplysninger om hvordan softwaren er konstrueret anvendes til at udlede testcases, f.eks. kode og design
- Omfanget af dækning af softwaren kan måles ud fra eksisterende testcases, og der kan systematisk udledes flere testcases for at øge dækningen

Almindelige egenskaber for erfaringsbaseret teknik omfatter:

- Personers kendskab og erfaring anvendes til at udlede testcases
- Testeres, udvikleres, brugeres og andre interessenters kendskab til softwaren, dens anvendelse og dets omgivelser
- Kendskab til sandsynlige defekter og deres fordeling

4.3 Specifikationsbaserede eller black-box-teknikker (K3)	150 minutter

Termer

Grænseværdianalyse, beslutningstabeltest, ækvivalenspartitionering, tilstandsovergangstest, usecase test.

4.3.1 Ækvivalenspartitionering (K3)

Input til softwaren eller systemet er opdelt i grupper, der forventes at have samme opførsel, så det er sandsynligt, at de bliver behandlet på samme måde. Ækvivalenspartitioner (eller klasser) kan findes for både gyldige og ugyldige data (dvs. værdier, der bør afvises). Man kan også identificere partitioner for output, interne værdier, tidsrelaterede værdier (f.eks. før eller efter en hændelse) og for grænsefladeparametre (f.eks. under integrationstest). Tests kan designes til at dække partitioner. Ækvivalenspartitionering (ÆP) kan anvendes på alle testniveauer.

Ækvivalenspartitionering kan anvendes som teknik til at opnå input- og outputdækning. Den kan anvendes til input fra mennesker, input via grænseflader til et system eller som grænsefladeparametre i integrationstest.

4.3.2 Grænseværdianalyse (K3)

Det er mest sandsynligt, at der opstår fejl netop på kanten af hver ækvivalenspartition, så grænseværdier er et område, hvor test kan forventes at afsløre defekter. Maksimums- og minimumsværdierne for en partition er dens grænseværdier. En grænseværdi for en gyldig partition er en gyldig grænseværdi. Grænseværdien for en ugyldig partition er en ugyldig grænseværdi. Der kan designes tests til at dække både gyldige og ugyldige grænseværdier. Når der designes testcases, vælges en test på hver grænseværdi.

Grænseværdianalyse kan anvendes på alle testniveauer. Den er relativt let at anvende, og den finder mange defekter. Derfor er detaljerede specifikationer et nyttigt redskab i planlægningen.

Denne teknik betragtes ofte som en udvidelse af ækvivalenspartitionering eller andre black-box-teknikker. Den kan anvendes på ækvivalensklasser for brugerinput på skærmen, såvel som på tidsgrænser (f.eks. time out, transaktionshastigheder mv.) eller tabelintervaller (f.eks. tabelstørrelsen er 256*256).

4.3.3 Test af beslutningstabel (K3)

Beslutningstabeller er en god måde at fange systemkrav, der indeholder logiske betingelser, og til at dokumentere det interne systemdesign. De kan anvendes til at registrere de komplekse forretningsregler, som et system skal implementere. Man danner en beslutningstabel ved at analysere specifikationen og identificere betingelser og systemhandlinger. Inputbetingelser og handlinger er ofte angivet på en sådan måde, at de enten er sande eller falske (Boolean). Beslutningstabellen indeholder også udløsende betingelser, ofte kombinationer af sand og falsk for alle inputbetingelser og deraf følgende handlinger for hver betingelseskombination. Hver

kolonne i tabellen kommer til at svare til en forretningsregel med en unik kombination af betingelser, der medfører afvikling af de handlinger, der hænger sammen med reglen. I en test med beslutningstabel-teknik skal man typisk have mindst én test pr. kolonne, hvad der vil give dækning for alle kombinationer af udløsende betingelser.

Styrken ved beslutningstabeltest er, at den danner betingelseskombinationer, der måske ellers ikke ville være blevet afviklet under test. Den kan anvendes i alle situationer, hvor softwarens reaktion afhænger af mere end én logisk beslutning.

4.3.4 Tilstandsovergangstest (K3)

Et system kan udvise forskellige reaktioner afhængig af aktuelle betingelser eller tidligere historie (dets tilstand). I dette tilfælde kan systemaspektet vises som et diagram over tilstandsovergange. Testeren kan betragte softwaren som tilstande, overgange mellem tilstande, input eller hændelser der udløser tilstandsændringer og handlinger, der kan være resultatet af disse overgange. Tilstandene i systemet eller genstanden under test er adskilte, kan identificeres og findes i begrænset antal. En tilstandstabel viser sammenhængen mellem tilstande og input og kan fremhæve mulige overgange, der er ugyldige. Tests kan designes til at dække en typisk rækkefølge af tilstande, til at dække hver eneste tilstand, til at aktivere alle overgange, til at aktivere specifikke rækkefølger af overgange, eller til at teste ugyldige overgange.

Tilstandsovergangstest anvendes meget i indlejret software og i teknisk automatisering i almindelighed. Teknikken er også velegnet til at modellere et forretningsobjekt, der har specifikke tilstande eller til test af skærm-dialog-flows (f.eks. internetprogrammer og forretningsscenarier).

4.3.5 Usecase test (K2)

Man kan udlede tests af usecases. En usecase beskriver samspillet mellem aktører (brugere eller systemer), der giver et resultat til en systembruger eller til kunden. Usecases kan være beskrevet på abstrakt niveau (forretnings-usecase, uden teknologibeskrivelse på forretningsproces-niveau) eller systemniveau (system-usecase på funktionelt niveau). Hver usecase har startbetingelser, der skal være opfyldt for at en usecase fungerer korrekt. Hver usecase afsluttes med slutbetingelser, der er de observérbare resultater og den endelige systemtilstand efter at usecasen er fuldført. En usecase har sædvanligvis et almindeligt (dvs. mest sandsynligt) scenarie og nogle gange desuden alternative scenarier.

Usecases beskriver procesflowet gennem et system baseret på systemets sandsynlige, faktiske anvendelse. Testcases, der tager udgangspunkt i usecases, er de bedste til at afdække defekter i realistiske proces-flows. Usecases, der ofte refereres til som scenarier, er meget nyttige til design af accepttests med deltagelse af kunde eller bruger. De hjælper også med til at afdække integrationsdefekter, der er forårsaget af samspil og interferens mellem forskellige komponenter. Det er ofte defekter, som en individuel komponenttest ikke kan afsløre.

4.4 Strukturbaserede eller white-box-teknikker (K4)	60 minutter

Termer

Kodedækning, beslutningsdækning, instruktionsdækning, strukturbaseret test .

Baggrund

Strukturbaseret test eller white-box test er baseret på en identificeret struktur i softwaren eller systemet, som det fremgår af følgende eksempler:

- Komponentniveau: Strukturen i en software-komponent, dvs. instruktioner, beslutninger, forgreninger og eventuelt også specificerede stier
- Integrationsniveau: Strukturen kan være et kaldetræ (et diagram, hvor moduler kalder andre moduler)
- Systemniveau: Strukturen kan være en menustruktur, forretningsproces- eller en websidestruktur

I dette afsnit behandles to koderelaterede strukturteknikker til kodedækning baseret på instruktioner og beslutninger. Til beslutningstest kan der anvendes et kontrolflow-diagram til at visualisere alternativer for hver beslutning.

4.4.1 Instruktionstest og -dækning (K4)

Ved komponenttest er instruktionsdækning en fastlæggelse af den procentdel af eksekvérbare instruktioner, der er aktiveret af et testcasesæt. Instruktionstest udleder testcases til afvikling af specifikke instruktioner, normalt for at øge instruktionsdækningen.

Instruktionsdækning er bestemt af hvor stort et antal af eksekvérbare instruktioner der er omfattet af (designede eller afviklede) testcases, divideret med antallet af alle eksekvérbare instruktioner i den kode der er omfattet af testen.

4.4.2 Beslutningstest og dækning (K4)

Beslutningsdækning, der er relateret til forgreningstest, er fastlæggelse af procentdelen af beslutningsresultater (f.eks. Sand og Falsk mulighederne i en IF-instruktion), der er aktiveret af testcasesæt. Med beslutningstest kan man udlede testcases til afvikling af specifikke beslutningsresultater. Forgreninger stammer fra beslutningspunkter i koden og viser overførslen af kontrol til forskellige steder i koden.

Beslutningsdækningen er bestemt af hvor mange beslutningsresultater, der er omfattet af (designede eller afviklede) testcases, divideret med antallet af samtlige, mulige udfald af beslutningerne i den kode, der er omfattet af testen.

Beslutningstest er en form for kontrolflow-test, da den genererer et specifikt kontrolflow gennem beslutningspunkterne. Beslutningsdækning er stærkere end instruktionsdækning: 100 % beslutningsdækning garanterer 100 % instruktionsdækning, men ikke vice versa.

4.4.3 Andre strukturbaserede teknikker (K1)

Der findes stærkere niveauer af strukturel dækning ud over beslutningsdækning, f.eks. betingelsesdækning og multibetingelsesdækning.

Dækningskonceptet kan også anvendes på andre testniveauer (f.eks. på integrationsniveau), hvor procentdelen af moduler, komponenter og klasser, der er blevet aktiveret af et testcasesæt, kan udtrykkes som modul-, komponent- eller klassedækning.

Det er nyttigt med værktøjsstøtte til strukturel test af kode.

4.5 Erfaringsbaserede teknikker (K2)	30 minutter

Termer

Udforskende test, fejlangreb.

Baggrund

Erfaringsbaseret test er test, som udledes ud fra testernes evner, intuition og erfaring med lignende programmer og teknologier. Sådanne teknikker kan være nyttige til at øge værdien at de systematiske teknikker, bl.a. ved at identificere specielle tests, der ikke så let fanges af formelle teknikker. Det gælder især når erfaringsbaseret test anvendes efter mere formelle tilgange. Imidlertid kan effektiviteten af denne teknik variere stærkt, afhængigt af testernes erfaringer.

En almindeligt brugt erfaringsbaseret teknik er fejlgætning. Almindeligvis forudser testere fejl på baggrund af deres erfaring. En struktureret tilgang til fejlgætningsteknikken er at opstille en liste over mulige fejl og over hvilke tests, der angriber disse fejl. Denne systematiske tilgang kaldes fejlangreb. Disse lister over defekter og afvigelser kan opbygges på basis af erfaring, tilgængelige defekt- og afvigelsesdata og ud fra almindeligt kendskab til, hvor der opstår fejl i software.

Udforskende test er samtidig testdesign, testafvikling, testlogning og læring, baseret på et testcharter, der indeholder testformål, og den udføres i tidsbokse. Det er en tilgang, der er mest nyttig, hvor der kun er få eller utilstrækkelige specifikationer og et stort tidspres. Udforskende test kan øge værdien af eller komplementere mere formel test. Den kan tjene som kontrol af testprocessen for at hjælpe med at sikre, at man finder de mest alvorlige defekter.

4.6 Valg af testteknikker (K2)	15 minutter

Termer

Ingen specifikke termer.

Baggrund

Valget af hvilke testteknikker, der skal anvendes, afhænger af forskellige faktorer, herunder systemtypen, regulerede standarder, kunde- eller kontraktkrav, risikoniveau, risikotype, testformål, tilgængelig dokumentation, testernes kendskab, tid og budget, udviklingslivscyklus, usecase-modeller og tidligere erfaring med fundne defekttyper.

Nogle teknikker er bedst i bestemte situationer og til bestemte testniveauer, andre kan anvendes på alle testniveauer.

Ved udarbejdelsen af testcases gør testerne hovedsageligt brug af en kombination af forskellige testteknikker som omfatter proces, regler og datadrevne teknikker for at sikre en passende dækning af testobjektet.

Referencer

4.1 Craig, 2002, Hetzel, 1988, IEEE 829-1998
4.2 Beizer, 1990, Copeland, 2004
4.3.1 Copeland, 2004, Myers, 1979
4.3.2 Copeland, 2004, Myers, 1979
4.3.3 Beizer, 1990, Copeland, 2004
4.3.4 Beizer, 1990, Copeland, 2004
4.3.5 Copeland, 2004
4.4.3 Beizer, 1990, Copeland, 2004
4.5 Kaner, 2002
4.6 Beizer, 1990, Copeland, 2004

5. Teststyring (K3)	170 minutter

Undervisningsmål for teststyring
Formålene viser, hvad d u skal kunne efter fuldførelse af hvert enkelt modul.

5.1. Testorganisation (K2)

LO-5.1.1 Kende til vigtigheden af uafhængig test. (K1)
LO-5.1.2 Forklare fordele og ulemper ved uafhængig test i en organisation. (K2)
LO-5.1.3 Kende de forskellige teammedlemmer, der skal tages i betragtning ved oprettelse af et testteam. (K1).
LO-5.1.4 Huske typiske testleder- og testeropgaver. (K1)

5.2. Testplanlægning og estimering (K3).

LO-5.2.1 Kende de forskellige niveauer og formål for testplanlægning. (K1)
LO-5.2.2 Opsummere formål og indhold af testplanen, testdesign-specifikationen og testproceduredokumenter i henhold til "Standard for Software Test Dokumentation" (IEEE 829-1998). (K2)
LO-5.2.3 Skelne mellem konceptuelt forskellige testmetoder, så som analytisk, modelbaseret, metodisk, proces-standard-overensstemmende, dynamisk/heuristisk, konsulterende og regressions-afvigende. (K2)
LO-5.2.4 Skelne mellem grunden til testplanlægning for et system og til at planlægge testafvikling. (K2)
LO-5.2.5 Skrive en testafviklingsplan for et givent sæt af testcases, når der tages hensyn til prioritering og tekniske og logistiske afhængigheder. (K3)
LO-5.2.6 Liste testforberedelses- og testafviklingsaktiviteter, som skal overvejes under testplanlægning. (K1)
LO-5.2.7 Huske typiske faktorer, der påvirker indsatsen vedrørende test. (K1)
LO-5.2.8 Skelne mellem to begrebsmæssigt forskellige estimeringsmetoder: Den metrikbaserede metode og den ekspertbaserede metode. (K2)
LO-5.2.9 Kende/redegøre for passende start- og slutkriterier for specifikke testniveauer og grupper af testcases (f.eks. integrationstest, accepttest eller testcases til brugervenlighedstest) (K2)

5.3. Overvågning og kontrol af testfremdrift (K2)

LO-5.3.1 Huske almindelige metrikker, der anvendes til overvågning af testforberedelse og afvikling. (K1).
LO-5.3.2 Forklare og sammenligne testmetrikker til testrapportering og testkontrol (f.eks. de defekter, der er fundet og rettet, og tests der er bestået/fejlet) i forhold til formål og anvendelse. (K2)
LO-5.3.3 Opsummere formål og indhold af testopsummeringsrapportdokument i henhold til "Standard for Software Test Dokumentation" (IEEE 829-1998) (K2)

5.4. Konfigurationsstyring (K2)

LO-5.4.1 Opsummere hvordan konfigurationsstyring støtter test. (K2)

5.5 Risiko og test (K2)

LO-5.5.1 Beskrive en risiko som et muligt problem, der kunne true gennemførelsen af et eller flere af interessenternes projektformål. (K2)
LO-5.5.2 Huske, at risici bestemmes af sandsynlighed (for at det sker) og effekt (den skade, der sker, hvis det sker). (K1)
LO-5.5.3 Skelne mellem projekt- og produktrisici. (K2)
LO-5.5.4 Kende typiske produkt- og projektrisici. (K1)
LO-5.5.5 Beskrive, ved hjælp af eksempler, hvordan risikoanalyse og risikostyring kan anvendes ved testplanlægning. (K2)

5.6 Hændelseshåndtering (K3)

LO-5.6.1 Kende indholdet af en hændelsesrapport i forhold til "Standard for Software Test Dokumentation" (IEEE 829). (K1)
LO-5.6.2 Skrive en hændelsesrapport, der dækker observationen af en afvigelse under test. (K3)

5.1 Testorganisation (K2)	30 minutter

Termer

Tester, testleder, testansvarlig.

5.1.1 Testorganisation og uafhængighed (K2)

Effektiviteten for at finde defekter ved test og reviews kan forbedres ved at anvende uafhængige testere. Muligheder for uafhængighed er:

- Ingen uafhængige testere. Udviklere tester deres egen kode.
- Uafhængige testere i udviklingsteams
- Uafhængigt testteam eller testgruppe i organisationen, der rapporterer til projektstyring eller højere ledelse
- Uafhængige testere fra virksomhedsorganisationen eller brugergruppen
- Uafhængige testspecialister til specifikke testformål, som f.eks. brugervenlighedstestere, sikkerhedstestere eller certificerende testere (der godkender et softwareprodukt i forhold til standarder og lovbestemmelser)
- Uafhængige testere, der er outsourcet eller eksterne i forhold til organisationen

Ved store, komplekse, sikkerhedskritiske projekter er det sædvanligvis bedst at have flere testniveauer, hvor nogle eller alle niveauer testes af uafhængige testere. Udviklingsmedarbejdere kan deltage i test, især på de lavere niveauer, men deres manglende objektivitet begrænser ofte deres effektivitet. Uafhængige testere kan have autoritet til at kræve og definere testprocesser og regler, men testere bør kun påtage sig sådanne procesrelaterede roller, når der er klart styringsmandat til at gøre det.

Fordelene ved uafhængighed omfatter:

- Uafhængige testere ser andre og anderledes defekter og er uvildige
- En uafhængig tester kan verificere de antagelser, andre har gjort under specifikation og implementering af systemet

Ulemper omfatter:

- Isolation fra udviklingsteamet (hvis de behandles som fuldstændig uafhængige)
- Udviklere kan miste ansvarsfølelsen for kvaliteten i produktet
- Uafhængige testere kan blive opfattet som en flaskehals eller få skylden for forsinkelsen af releases

Testopgaver kan udføres af personer med en specifik testrolle eller kan udføres af nogen med en anden rolle, som f.eks. projektleder, kvalitetsansvarlig, udvikler, forretnings- og domæneekspert, infrastruktur eller it-drift.

5.1.2 Testlederes og testeres opgaver (K1)

Der er to testroller: testleder og tester. De aktiviteter og opgaver, der udføres af personer i disse to roller afhænger af projekt- og produktkonteksten, personerne der udfylder rollerne, og organisationen.

Nogle gange kaldes testlederen testansvarlig eller testkoordinator. Rollen som testleder kan udføres af en projektleder, en udviklingschef, en ansvarlig for kvalitetssikring eller en leder af en testgruppe. I større projekter kan der være to stillinger: testleder og testansvarlig. Typisk planlægger, overvåger og kontrollerer testlederen de testaktiviteter og opgaver, som er defineret i afsnit 1.4.

Testlederens opgaver kan typisk omfatte:

- At koordinere teststrategi og testplan med projektledere og andre
- At udarbejde og vedligeholde en teststrategi for projektet og en testpolitik for organisationen
- At bidrage med testperspektivet ved andre projektaktiviteter, f.eks. ved integrationsplanlægning
- At planlægge tests – under overvejelse af konteksten og forståelse af testformål og risici – herunder valg af testtilgang, estimering af tid, indsats og udgifter for testen, fremskaffelse af ressourcer, definition af testniveauer, cyklusser og planlægning af hændelseshåndtering
- At starte specifikation, forberedelse, implementering og afvikling af tests, overvåge testresultater og kontrollere slutkriterier
- At tilpasse planlægningen baseret på testresultater og fremdrift (nogle gange dokumenteret i statusrapporter) og at prioritere ved problemer
- At opsætte passende konfigurationsstyring af test-delprodukter med henblik på sporbarhed
- At introducere egnede metrik-systemer til måling af testfremdrift og evaluering af kvaliteten af test og produkt
- At beslutte hvad der skal automatiseres, i hvilken grad og hvordan
- At vælge værktøjer til at støtte test og organisere eventuel uddannelse i brug af værktøj for testere
- At træffe beslutning om implementering af testmiljø
- At skrive testopsummeringsrapporter baseret på de oplysninger, der indsamles under testen

Testerens opgaver kan typisk omfatte:

- At reviewe og bidrage til testplaner
- At analysere, reviewe og vurdere brugerkrav, specifikationer og modeller for testbarhed
- At fremstille testspecifikationer
- At opsætte testmiljø (ofte koordineret med systemadministrationen og netværksstyring)
- At forberede og fremskaffe testdata
- At implementere tests på alle testniveauer, afvikle og registrere tests, evaluere resultater og dokumentere afvigelser fra forventede resultater
- At bruge testadministrations- eller styringsværktøjer og testovervågningsværktøjer efter behov
- At automatisere tests (kan være understøttet af en udvikler eller en testautomatiseringsekspert)
- At måle performance af komponenter og systemer, hvis nødvendigt
- At reviewe tests, der er udviklet af andre

Personer, der arbejder med testanalyse, testdesign, specifikke testtyper eller testautomatisering kan være specialister i disse roller. Afhængig af testniveauet og de risici, der relaterer sig til produktet og projektet, kan disse overtage rollen som tester, idet der bevares en grad af uafhængighed. Typiske vil testere på komponent- og integrationsniveau være udviklere, testere på accepttestniveau være forretningseksperter og brugere, og testere til driftsaccepttest være operatører.

5.2 Testplanlægning og estimering (K3)	40 minutter

Termer

Testtilgang

5.2.1 Testplanlægning (K2)

Dette afsnit beskriver formålet med testplanlægning i udviklings- og implementeringsprojekter og for vedligeholdelsesaktiviteter. Planlægning kan være dokumenteret i en hovedtestplan og i separate testplaner for testniveauer, som f.eks. systemtest og accepttest. Oversigt over testplanlægningsdokumenter er givet i "Standard for Software Test Dokumentation" (IEEE 829).

Planlægning er påvirket af organisationens testpolitik, omfanget af test, formål, risici, begrænsninger, alvorlighed, testbarhed og tilgængelighed af ressourcer. Jo mere projekt- og test-planlægningen skrider frem, jo mere information vil være tilgængelig, og jo flere detaljer kan man inkludere i planen.

Testplanlægning er en løbende aktivitet, og den udføres i alle processer og aktiviteter i softwarens livscyklus. Feedback fra testaktiviteter anvendes til at erkende ændrede risici, så man kan justere testplanen.

5.2.2 Testplanlægningsaktiviteter (K3)

Testplanlægningsaktiviteter for et helt eller dele af et system kan indeholde:

- Vurdering af omfang og risiko og identifikation af formålene med test
- Definition af den overordnede testmetode (teststrategi), inklusive testniveauer, start- og slutkriterier
- Integration og koordination af testaktiviteter i softwarens livscyklus: anskaffelse, levering, udvikling, drift og vedligeholdelse
- Beslutning om hvad der skal testes, hvem der skal teste, hvordan testaktiviteterne skal udføres, og hvordan testresultater skal evalueres
- Tidsforbrug til testanalyse og designaktiviteter
- Tidsforbrug til testimplementering, afvikling og evaluering
- Tildeling af ressourcer til de forskellige definerede aktiviteter
- Planlægge af testdokumentation: Mængde, detaljeniveau, struktur og skabelon
- Valg af metrikker til overvågning og kontrol af testforberedelse og afvikling, defekt-løsning og risikomomenter
- Fastsættelse af detaljeniveauet for testprocedurer for at kunne give oplysninger nok til at understøtte reproducerbar testforberedelse og afvikling

5.2.3 Startkriterier (K2)

Startkriterierne beskriver hvornår testen starter, f.eks. ved starten af et testniveau eller når et set af testcases er færdige til at blive udført

Typiske startkriterier kan omfatte:

- Testmiljøets færdighedsgrad og tilgængelighed

- Testværktøjers beredskab i testmiljøet
- Tilgængelighed af testbar kode
- Tilgængelighed af testdata

5.2.4 Slutkriterier (K2)

Formålet med slutkriterier er at definere, hvornår testen skal stoppes, som f.eks. ved afslutning af et testniveau, eller når et testsæt har nået et specifikt mål.

Typiske slutkriterier kan bestå af:

- Måling af grundighed, som f.eks. kode-, funktionalitets- og risikodækning
- Vurdering af defekttæthed eller pålidelighedsmålinger
- Udgifter
- Restrisici som f.eks. defekter, der ikke er rettet eller manglende testdækning i visse områder
- Tidsplaner så som sådanne, som er baseret på time-to-market

5.2.5 Testestimering (K2)

To metoder til testestimering:

- Den metrikbaserede tilgang: Estimering af testindsats baseret på metrikker fra tidligere eller lignende projekter eller baseret på typiske værdier
- Den ekspertbaserede tilgang: Estimering af opgaver af ejeren af disse opgaver eller af eksperter

Når man har estimeret testindsatsen, kan man også identificere de nødvendige ressourcer og lave en tidsplan.

Testindsatsen kan være afhængig af en række faktorer, herunder:

- Karakteristika for produktet: kvaliteten af specifikationen og andre oplysninger, der anvendes til testmodeller (dvs. testgrundlag), produktstørrelsen, kompleksiteten i problemområdet, kravene til pålidelighed og sikkerhed og kravene til dokumentation
- Karakteristika for udviklingsprocessen: organisationens stabilitet, anvendte værktøjer, testproces, de involverede personers evner og tidspres
- Testresultatet: antal defekter og mængden af nødvendig genbearbejdning

5.2.6 Testtilgang, teststrategier (K2)

I testtilgangen ligger implementeringen af teststrategien for et specifikt projekt. Testtilgangen defineres og finjusteres i testplaner og testdesign. Den omfatter typisk beslutninger der hviler på (test-)projektets mål og risikovurdering. Den er udgangspunkt for planlægning af testprocessen, for udvælgelsen af testdesignteknikker og testtyper, samt for fastlæggelse af start- og slutkriterierne.

Testtilgangen er afhængig af konteksten og kan omfatte overvejelser om risici, farer og sikkerhed, tilgængelige ressourcer og kompetencer, teknologi, systemets natur (f.eks. egenudvikling versus hyldesoftvare), testformål og lovbestemmelser.

Typiske metoder eller strategier indeholder:

- Analytiske metoder, som f.eks. risikobaseret test, hvor test rettes mod områder, hvor der er den største risiko
- Modelbaserede metoder, som f.eks. stokastisk test vha. statistiske oplysninger om afvigelseshyppigheder (som f.eks. pålideligheds-vækstmodeller) eller anvendelse (som f.eks. driftsprofiler)
- Metodiske tilgange, som f.eks. baseret på afvigelser (herunder fejlgætning og fejlangreb), erfaringer, tjeklister og kvalitetsegenskaber
- Proces- og standarddefinerede metoder, f.eks. som angivet af industrispecifikke standarder og forskellige agile metoder
- Dynamiske og heuristiske metoder, som f.eks. udforskende test, hvor test er mere reaktiv i forhold til hændelser end planlagt, og hvor afvikling og vurdering er sideløbende opgaver
- Konsultative metoder, som f.eks. de hvor testdækning primært drives af råd og vejledning fra teknologi- og/eller forretningsdomæneeksperter uden for testteamet
- Regressionsmodsætningsmetoder, som f.eks. de der indeholder genbrug af eksisterende testmateriale, omfattende automatisering af funktionelle regressionstests og standardtestsæt

De forskellige metoder kan kombineres, f.eks. til en risikobaseret, dynamisk metode.

Valget af en testtilgang bør omfatte konteksten, herunder:

- Risiko for projektets fiasko og risiko for produktafvigelse for mennesker, miljø og virksomhed
- Medarbejdernes evner og erfaringer i de foreslåede teknikker, værktøjer og metoder
- Formålet med testopgaven og testteamets mission
- Regelbestemte aspekter, som f.eks. eksterne og interne bestemmelser for udviklingsprocessen
- Produktets og forretningens natur

5.3 Overvågning og kontrol af testfremdriften (K2)	20 minutter

Termer

Fejltæthed, afvigelseshyppighed, testkontrol, testovervågning, testrapport.

5.3.1 Overvågning af testfremdrift (K1)

Formålet med testovervågning er at give feedback og synlighed af testaktiviteterne. De oplysninger, der skal overvåges, kan indsamles manuelt eller automatisk og kan anvendes til at måle slutkriterier, som f.eks. dækning. Metrikker kan også anvendes til at vurdere fremdriften i forhold til tidsplanen og budgettet. Almindelige testmetrikker omfatter:

- Procentdelen af arbejde, der er brugt til testcaseforberedelse (eller procentdelen af planlagte testcases der er forberedt)
- Procentdelen af arbejde, der er brugt til testmiljøforberedelse
- Afvikling af testcase (f.eks. antal testcases, der er kørt/ikke kørt, og testcases, der har bestået/fejlet)
- Defektoplysninger (f.eks. defekttæthed, defekter, der er fundet og rettet, afvigelseshyppighed og resultater af gentest)
- Testdækning af krav, risici eller kode
- Testernes subjektive tillid til produktet
- Datoer for testmilepæle
- Testudgifter, herunder udgifterne sammenlignet med fordelene ved at finde den næste defekt eller at køre den næste test

5.3.2 Testrapportering (K2)

Testrapportering vedrører opsummering af oplysninger om testopgaven, herunder:

- Hvad der skete i testperioden, som f.eks. datoer, hvor slutkriterierne blev opfyldt
- Analyseret information og metrikker til at støtte anbefalinger og beslutninger om fremtidige handlinger, som f.eks. en vurdering af tilbageværende defekter, det økonomiske udbytte ved fortsat test, udestående risici, og tillidsniveauet for den testede software

Oversigten over en testopsummeringsrapport er givet i "Standard for Software Test Dokumentation" (IEEE 829-1998).

Metrikker bør indsamles under og ved afslutningen af et testniveau for at vurdere:

- Tilstrækkelighed af testformålene for det givne testniveau
- Tilstrækkelighed af de udførte testtilgange
- Effektiviteten af testen i forhold til dens formål

5.3.3 Testkontrol (K2)

Testkontrol beskriver vejledning og rettelseshandlinger som et resultat af de oplysninger og de metrikker, der er indsamlet og rapporteret. Handlinger kan dække enhver testaktivitet og kan påvirke enhver anden aktivitet eller opgave.

Eksempler på testkontrolhandlinger er:

- Træffe beslutninger baseret på information fra testovervågning
- Omprioritere tests, når en identificeret risiko opstår (f.eks. ved sen levering af software)
- Ændre testtidsplanen på grund af (manglende) tilgængelighed af et testmiljø
- Sætte et indgangskriterium, f.eks. med krav om at rettelser gentestes af en udvikler, før de accepteres i et build

5.4 Konfigurationsstyring (K2)	10 minutter

Termer

Konfigurationsstyring, versionsstyring.

Baggrund

Formålet med konfigurationsstyring er at etablere og fastholde produktintegriteten (komponenter, data og dokumentation) for softwaren eller systemet igennem hele projektets og produktets livscyklus.

For test kan konfigurationsstyring involvere sikring af, at:

- Alle test-delprodukter er identificeret, versionsstyret, sporet for ændringer, relateret til hinanden og relateret til udviklingselementer (testobjekter), så man kan fastholde sporbarheden igennem testprocessen
- Alle identificerede dokumenter og softwareelementer refereres entydigt i testdokumentationen

Konfigurationsstyring hjælper testerne til entydigt at identificere (og til at gendanne) det testede element, testdokumenter, tests og testharness.

Under testplanlægning bør konfigurationsstyringsprocedurer og infrastruktur (værktøjer) vælges, dokumenteres og implementeres.

5.5 Risiko og test (K2)	30 minutter

Termer

Produktrisiko, projektrisiko, risiko, risikobaseret test.

Baggrund

Risiko kan defineres som muligheden for at en hændelse, fare, trussel eller situation opstår og dens uønskede konsekvenser, et muligt problem. Risikoniveauet bestemmes af sandsynligheden for, at der opstår en uheldig hændelse og effekten heraf (den skade, der opstår som følge af hændelsen).

5.5.1. Projektrisici (K2)

Projektrisici er risici, der omfatter projektets evne til at leve op til sit formål, som f.eks.:

- Organisatoriske faktorer:
  - mangel på kompetencer, færdigheder og personale
  - medarbejderproblemer
  - politiske emner, som f.eks.
    - testeres problemer med at kommunikere deres behov og testresultater
    - manglende opfølgning på oplysninger, der er fundet ved test og reviews (f.eks. ingen forbedring af udviklings- og testpraksis)
  - forkert attitude imod eller forventninger til test (f.eks. ikke værdsættelse af værdien af at finde defekter under test)
- Tekniske emner:
  - problemer med definition af kravene
  - krav, som projektet ikke kan levere under de givne begrænsninger
  - testmiljø ikke klar til tiden
  - Forsinket datakonvertering, migrationsplanlægning eller udvikling og test af værktøjer til data-konvertering/migration
  - lav kvalitet af design, kode, konfigurationsdata, testdata og tests
- Leverandøranliggender:
  - tredjeparts afvigelser
  - kontraktuelle anliggender

Når disse risici analyseres, styres og afhjælpes, skal den testansvarlige følge veletablerede projektstyringsprincipper. Afsnittet vedr. testplaner i "Standard for Software Test Dokumentation" (IEEE 829-1998) kræver, at der angives risici og alternative planer.

5.5.2. Produktrisici (K2)

Mulige afvigelsesområder (uheldige fremtidige hændelser eller farer) i softwaren eller systemet er kendt som produktrisici, da de udgør en risiko for produktkvaliteten, som f.eks.

- Levering af afvigelsebehæftet software
- Risiko for, at software og hardware kan medføre skade på personer og virksomheder
- Ringe softwareegenskaber (f.eks. funktionalitet, pålidelighed, brugervenlighed og performance)

- Dårlig dataintegritet og –kvalitet (f.eks. problemer med migrering, konvertering og transport af data eller brud på datastandarder)
- Software der ikke udfører, hvad der var hensigten

Risici anvendes til at bestemme, hvor test skal begynde, og hvor der skal testes yderligere; test anvendes til at reducere risikoen for, at en uheldig effekt opstår, eller for at reducere effekten af en uheldig reaktion.

Produktrisici er en speciel risikotype der kan afgøre om et projekt bliver en succes. Test kan begrænse risikoen. Det sker ved at opgøre projektets resterende risiko som en måling af effektiviteten til at fjerne kritiske defekter og som en vurdering af alternative planers effektivitet.

En risikobaseret testmetode giver proaktiv lejlighed til at reducere produktrisikoniveauer, og den kan og bør starte i begyndelsen af et projekt. Den involverer identifikation af produktrisici og brugen af dem til at vejlede om testplanlægning og kontrol, specifikation og afvikling af tests. I en risikobaseret metode kan de identificerede risici anvendes til:

- At bestemme de testteknikker, der skal anvendes
- At bestemme omfanget af den test, der skal udføres
- At prioritere test i et forsøg på at finde kritiske defekter så tidligt som muligt
- At bestemme om der kan anvendes nogen ikke-test-aktiviteter til at reducere risikoen (f.eks. ved at uddanne uerfarne designere)

Risikodrevet test trækker på interessenternes samlede viden og indsigt til at bestemme risici og de niveauer af test, der er nødvendige for at håndtere disse risici.

I sikringen af at risikoen for en produktafvigelse minimeres, giver risikostyringsaktiviteter en disciplineret metode til at:

- Vurdere (og regelmæssigt revurdere) hvad der kan gå galt (risici)
- At bestemme hvilke risici, der er vigtigst at håndtere
- At implementere handlinger, der skal håndtere disse risici

Endvidere kan test give støtte til identifikation af nye risici, kan hjælpe med at bestemme hvilke risici, der bør reduceres og kan sænke usikkerhed omkring risici.

5.6 Hændelseshåndtering (K3)	40 minutter

Termer

Hændelsesregistrering, hændelseshåndtering, hændelsesrapport.

Baggrund

Eftersom et af formålene med test er at finde defekter, skal uoverensstemmelser mellem aktuelle og forventede resultater registreres som hændelser. En hændelse bør undersøges og kan herefter vise sig at være en defekt. Det bør være fastlagt hvorledes der skal tages hånd om hændelser og fejl. Hændelser bør følges fra opdagelse og klassifikation til rettelse og bekræftelse af løsningen. For at håndtere alle hændelser til afslutning bør en organisation etablere en proces og regler for klassifikation.

Hændelser kan opstå under udvikling, review, test eller ved anvendelse af et softwareprodukt. De kan rejses for problemer i kode, i det fungerende system eller i enhver dokumenttype. Dokumenterne omfatter krav, udviklingsdokumenter, testdokumenter, og brugeroplysninger, som f.eks. "Hjælp" og installationsvejledninger.

Hændelsesrapporter har følgende formål:

- At give udviklere og andre parter feedback om problemet for at muliggøre identifikation, isolation og rettelse efter behov
- At give testledere et middel til at følge kvaliteten af systemet under test og testens fremdrift
- At give ideer til testprocesforbedring

Detaljer i hændelsesrapporten kan omfatte:

- Dato for udstedelse, udstedende organisation og forfatter
- Forventede og faktiske resultater
- Identifikation af testelement, (konfigurationselement) og miljø
- Software- eller systemlivscyklusproces i hvilken hændelsen blev observeret
- Beskrivelse af hændelsen for at gøre det muligt at gentage den og finde en løsning, inklusive logfiler, databasedumps eller skærmprint.
- Omfang eller grad af påvirkning af interessentens interesser
- Alvorlighed af påvirkning af systemet
- Vigtighed/prioritet for rettelse
- Hændelsesstatus (f.eks. åben, i bero, kopi, venter på at blive rettet, rettelse afventer gentest, lukket)
- Konklusioner, anbefalinger og godkendelser
- Globale problemer, som f.eks. andre områder, der kan blive påvirket af en ændring som følge af hændelsen
- Ændringshistorie, som f.eks. rækkefølgen af handlinger, der udføres af projektets deltagere vedrørende hændelsen for at isolere, reparere og bekræfte den som rettet
- Referencer, indeholdende identiteten af den testcasespecifikation, som afdækkede problemet.

Strukturen på en hændelsesrapport er også beskrevet i "Standard for Software Test Dokumentation" (IEEE 829-1998) og er her benævnt en uregelmæssighedsrapport.

Referencer

5.1.1 Black, 2001, Hetzel, 1988
5.1.2 Black, 2001, Hetzel, 1988
5.2.5 Black, 2001, Craig, 2002, IEEE 829, Kaner 2002
5.3.3 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE 829-1998
5.4 Craig, 2002
5.5.2 Black, 2001, IEEE 829-1998
5.6 Black, 2001, IEEE 829-1998

6. Værktøjsstøtte til test (K2)	80 minutter

Undervisningsmål for værktøjssupport til test
Formålene viser, hvad du skal kunne efter fuldførelse af hvert enkelt modul.

6.1. Typer af testværktøjer (K2)

LO-6.1.1 Klassificere forskellige typer af testværktøjer i forhold til testprocesaktiviteterne. (K2)
LO-6.1.3 Forklare termen testværktøj og formålet med værktøjssupport i test. (K2)[2]

6.2. Effektiv brug af værktøjer: Mulige fordele og risici (K2)

LO-6.2.1 Opsummere mulige fordele og risici ved testautomatisering og værktøjsstøtte til test. (K2)
LO-6.2.2 Huske de særlige hensyn til testafviklingsværktøjer, statisk analyse og teststyringsværktøjer. (K1)

6.3. Introduktion af værktøj i en organisation (K1)

LO-6.3.1 Anføre hovedprincipperne ved introduktion af et værktøj i en organisation. (K1)
LO-6.3.2 Anføre målene for en proof-of-concept for værktøjsvurdering og en pilotfase for værktøjsimplementering. (K1)
LO-6.3.3 Vide, at der kræves andet end blot erhvervelse af et værktøj til god værktøjsstøtte. (K1)

[2] LO-6.1.2 er bevidst sprunget over

6.1 Typer af testværktøjer (K2)	45 minutter

Termer

Konfigurationsstyringsværktøj, dækningsværktøj, debugging-værktøj, driver, dynamisk analyseværktøj, hændelseshåndteringsværktøj, loadtestværktøj, modelleringsværktøj, overvågningsværktøj, performancetestværktøj, undersøgelseseffekt, kravstyringsværktøj, reviewprocesstøtteværktøj, sikkerhedsværktøj, statisk analyseværktøj, stresstestværktøj, stub, testsammenligner, testdataforberedelsesværktøj, testdesignværktøj, testharness, testafviklingsværktøj, teststyringsværktøj, rammeværktøj for komponenttest.

6.1.1. Værktøjsstøtte til test (K2)

Der findes testværktøjer, der kan understøtte test:

1. Værktøjer der anvendes direkte i test som testafviklingsværktøjer, værktøjer til generering af testdata og værktøjer til resultatsammenligning
2. Værktøjer der hjælper med at forvalte testprocessen, testresultater, data, krav, hændelser, defekter osv., og som overvåger og rapporterer testafviklingen
3. Værktøjer der undersøger, f.eks. som overvåger filaktiviteten for en applikation
4. Ethvert værktøj der hjælper med test. I denne betydning kan f.eks. et regneark også være et testværktøj.

Værktøjsstøtte til test kan have et eller flere af følgende formål, afhængigt af sammenhængen:

- At forbedre effektiviteten af testaktiviteter ved at automatisere gentagne opgaver eller ved at støtte manuelle testaktiviteter som testplanlægning, testdesign, testrapportering og overvågning
- At automatisere aktiviteter der kræver betydelige ressourcer, hvis de skal gøres manuelt, f.eks. statisk test
- At automatisere aktiviteter der ikke kan udføres manuelt (f.eks. performancetest af client-server applikationer i stor skala)
- At øge pålideligheden af test, ved f.eks. at automatisere store datasammenligninger eller ved at simulere adfærd

Begrebet "testrammer" er ofte anvendt i branchen i mindst tre betydninger:

- Genanvendelige og udvidelsesvenlige testbiblioteker, der kan bruges til at bygge testværktøjer (også kaldet teststilladser)
- En type af design af testautomatisering (f.eks. datastyret eller nøgleordsstyret)
- Den overordnede proces for afvikling af test

I dette pensum er termen "testrammer" anvendt i de første to betydninger, som beskrevet i afsnit 6.1.6.

6.1.2. Klassificering af testværktøjer (K2)

Der findes en række værktøjer, der understøtter forskellige aspekter ved test. Værktøjerne kan klassificeres på basis af flere kriterier såsom formål, kommercielle / gratis / open-source /

shareware, den anvendte teknologi og så videre. Nedenfor er værktøjerne klassificeret i forhold til de testaktiviteter, de støtter.

Nogle værktøjer støtter en enkelt aktivitet. Andre støtter flere og er klassificeret under deres hovedaktivitet. Værktøjer fra en enkelt leverandør, især dem, der er designet til at arbejde sammen, kan være samlet i én pakke.

Nogle typer testværktøjer kan være invaderende, hvor værktøjet påvirker det aktuelle testresultat. Aktuel timing kan f.eks. være forskellig, afhængig af hvordan man måler den med forskellige performanceværktøjer. Man får forskellig måling af kodedækning afhængigt af hvilket dækningsværktøj, man anvender. Virkningen af invaderende værktøjer kaldes undersøgelseseffekten.

Nogle værktøjer tilbyder støtte, der er mest egnet til udviklere (f.eks. under komponent- og komponentintegrationstest). Sådanne værktøjer er markeret med "(U)" i klassifikationerne nedenfor.

6.1.3. Værktøjssupport til styring af test (K1)

Styringsværktøjer anvendes i alle testaktiviteter gennem hele softwarens livscyklus.

Teststyringsværktøjer

Disse værktøjer giver grænseflader for testafvikling, fejlopfølgning og kravstyring, sammen med støtte til kvantitativ analyse og rapportering af testobjekter. De understøtter sporbarhed af testobjekter til kravspecifikationer og kan have en selvstændig funktion for versionsstyring eller en grænseflade til en ekstern versionsstyring.

Kravstyringsværktøjer

Disse værktøjer lagrer krav, lagrer attributter for krav (f.eks. prioritet), giver en unik identifikation og støtter sporbarhed af krav til individuelle tests. De kan også hjælpe med at identificere usammenhængende eller manglende krav.

Hændelseshåndteringsværktøjer (fejlopfølgningsværktøjer)

Disse værktøjer lagrer og styrer hændelsesrapporter, dvs. defekter, afvigelser, ændringsønsker, opfattede problemer og uregelmæssigheder og hjælper med at styre hændelsernes livscyklus, eventuelt med støtte til statistisk analyse.

Konfigurationsstyringsværktøjer

Disse værktøjer er ikke egentlige testværktøjer, men kan være nødvendige for lagring og versionsstyring af test-delprodukter og relateret software, især når man konfigurerer mere end ét hardware/software-miljø i form af styresystemversioner, oversættere, browsere, osv.

6.1.4 Værktøjsstøtte til statisk test (K1)

Statiske testværktøjer giver en omkostningseffektiv måde at finde defekter tidligt i udviklingsprocessen.

Reviewværktøjer

Disse værktøjer hjælper med reviewprocesser, tjeklister og review af retningslinjer. De bruges til at lagre og kommunikere reviewkommentarer og rapporter om fejl og indsats. De kan eventuelt understøtte online-reviews for store eller geografisk spredte teams.

Statiske analyseværktøjer (U)

Disse værktøjer hjælper udviklere og testere med at finde fejl før den dynamiske test ved at håndhæve kodestandarder (herunder sikker kodning) og ved at analysere strukturer og afhængigheder. De kan også hjælpe med planlægning og risikoanalyse ved at levere metrikker for koden (f.eks. kompleksitet).

Modelleringsværktøjer (U)

Disse værktøjer bruges til at validere softwaremodeller, f.eks. en fysisk datamodel for en relationel database, ved at vise uoverensstemmelser og finde defekter. Sådanne værktøjer kan ofte generere modelbaserede testcases.

6.1.5. Værktøjsstøtte til testspecifikation (K1)

Testdesignværktøjer

Disse værktøjer bruges til at generere testinputs eller eksekvérbare tests og/eller testorakler fra krav, grafiske brugergrænseflader, designmodeller (tilstand, data eller objekt) eller fra kode.

Testdataforberedelsesværktøjer (U)

Testdataforberedelsesværktøjer manipulerer databaser, filer eller dataoverførsler til opsætning af data, der skal anvendes under afvikling af tests for at sikre sikkerhed gennem dataanonymitet.

6.1.6. Værktøjsstøtte til testafvikling og -logning (K1)

Testafviklingsværktøjer

Testafviklingsværktøjer gør det muligt at afvikle tests automatisk eller halvautomatisk vha. lagrede inputs og forventede resultater. De bruger et scriptsprog og producerer normalt en log for hver testkørsel. De kan også optage tests. Scripts kan tilpasses ved hjælp af et scriptsprog eller en grafisk brugergrænseflade-baseret konfiguration for parameterstyring af data.

Testharness-/enhedstest-rammeværktøjer (U)

Testharness kan gøre det lettere at teste komponenter eller en del af et system ved at simulere det miljø hvor testobjektet skal køre gennem mock objects i form af stubbe eller drivere.

Testsammenlignere

Testsammenlignerne bestemmer forskelle mellem filer, databaser og testresultater. Testafviklingsværktøjer indeholder typisk dynamiske sammenlignere, men sammenligning efter afviklingen kan også udføres af et separat sammenligningsværktøj. En testsammenligner kan anvende et testorakel, specielt hvis det er automatiseret.

Dækningsmåleværktøjer (U)

Dækningsmåleværktøjer kan enten være invaderende eller ikke-invaderende og måler procentdelen af specifikke typer af kodestruktur, der er blevet aktiveret af et testsæt (f.eks. instruktioner, forgreninger, beslutninger og modul- eller funktionskald).

Sikkerhedsværktøjer

Disse værktøjer bruges til at vurdere sikkerhedsegenskaber ved software. Dette omfatter evaluering af softwarens evne til at beskytte datafortrolighed, integritet, ægthedsbevis, autorisation, tilgængelighed og uafviselighed. Sikkerhedsværktøjer er overvejende beregnet på en bestemt kombination af teknologi, platform og formål.

6.1.7. Værktøjsstøtte til performance og overvågning (K1)

Dynamiske analyseværktøjer (U)

Dynamiske analyseværktøjer finder defekter, der kun afsløres, når softwaren eksekveres, f.eks. defekter med tidsafhængigheder eller hukommelseslæk. Værktøjerne anvendes typisk ved komponent- og komponentintegrationstest og ved test af middleware.

Performancetest-/loadtest-/stresstestværktøjer

Performancetestværktøjer overvåger og rapporterer om, hvordan et system opfører sig under forskellige simulerede anvendelsesbetingelser i form af et antal samtidige brugere, deres ramp-up mønster, frekvens og relativ procentdel af transaktioner. Simuleringen af belastningen sker ved at skabe virtuelle brugere, der udfører et udvalgt sæt af transaktioner fordelt på forskellige testmaskiner (belastningsgeneratorer).

Overvågningsværktøjer

Overvågningsværktøjer overvåger systemet konstant. De verificerer og rapporterer om anvendelsen af specifikke systemressourcer og giver advarsler om mulige serviceproblemer.

6.1.8. Værktøjssupport til specifikke programområder (K1)

Vurdering af datakvalitet

Data er i centrum for f.eks. datakonverterings- og migrationsprojekter og applikationer som datawarehouses. De kan variere i vigtighed og volumen. I sådanne sammenhænge kan man bruge værktøjer til at vurdere datakvaliteten for at reviewe og verificere datakonvertering og de tilhørende migrationsregler for at sikre, at de behandlede data er korrekte, fuldstændige og i overensstemmelse med en foruddefineret standard.

Der findes også testværktøjer beregnet til brugervenlighedstest.

6.2 Effektiv brug af værktøjer: Mulige fordele og risici (K2)	20 minutter

Termer

Datastyret (test), nøgleordsstyret (test), scriptsprog.

6.2.1. Mulige fordele og risici ved værktøjsstøtte til test (K2)

Blot det at købe eller lease et værktøj garanterer ikke succes med værktøjet. Hver værktøjstype kan kræve ekstra indsats for at opnå varige fordele. Der er potentielle fordele ved anvendelse af værktøjer til test, men der er også risici.

Mulige fordele ved anvendelse af værktøjer omfatter:

- Reduktion af repetitivt arbejde (f.eks. kørsel af regressionstests, genindtastning af de samme testdata, og kontrol i forhold til kodestandarder)
- Større konsistens og repeterbarhed (f.eks. tests afviklet vha. et værktøj, og tests udledt fra krav)
- Objektiv vurdering (f.eks. statiske målinger, dækning)
- Let adgang til oplysninger om tests eller testafvikling (f.eks. statistikker og grafer om testfremdrift, hændelseshyppighed og performance)

Risici ved anvendelse af værktøjer omfatter:

- Urealistiske forventninger til værktøjet (herunder funktionalitet og let anvendelighed)
- Undervurdering af tid, udgifter og indsats for den første introduktion af et værktøj (herunder undervisning og ekstern ekspertise)
- Undervurdering af tid og den nødvendige indsats før man får varig fordel af værktøjet (f.eks. behov for ændringer i testprocessen og den måde, værktøjet anvendes på)
- Undervurdering af den nødvendige indsats til at vedligeholde testdelprodukter, der er genereret af værktøjet
- For stor tillid til værktøjet (automatiseret testdesign eller test, hvor den manuelle proces ville være bedre)
- At forsømme versionsstyring af test-delprodukter i værktøjet
- At overse problemer med relationer og tværoperationalitet mellem kritiske værktøjer såsom kravstyringsværktøj, versionsstyringsværktøj, hændelseshåndteringsværktøjer, fejlopfølgningsværktøjer samt værktøjer fra flere leverandører
- Risiko for at værktøjsleverandøren går ud af markedet, udfaser værktøjet eller sælger værktøjet til en anden leverandør
- Risiko for manglende support, opgraderinger og fejlrettelser
- Risiko for lukning af open-source og andre, gratis værktøjer
- Uforudsete risici såsom manglende evne til at understøtte en ny platform

6.2.2. Specielle overvejelser for nogle værktøjstyper (K1)

Testafviklingsværktøjer

Testafviklingsværktøjer tester objekter ved hjælp af automatiserede scripts. Denne værktøjstype kræver ofte en stor indsats for at give fordele.

Optagelse af tests ved at optage handlinger fra en manuel tester ser attraktivt ud, men denne metode kan ikke udvides til store antal automatiserede tests. Et optaget script er en lineær fremstilling med specifikke data og handlinger som en del af hvert script. Denne type scripts kan være ustabil, når der opstår uventede hændelser.

En datastyret metode udskiller data, f.eks. til et regneark. Det giver et mere generisk script, der kan udføre samme test med forskellige data. Testere kan indsætte testdata til disse foruddefinerede scripts, også uden at kende scriptsproget.

Der findes også værktøjer, der i stedet for at bruge hårdt kodede datakombinationer fra et regneark genererer data under afviklingen. For eksempel kan et værktøj bruge en algoritme, der genererer en tilfældig bruger-id.

Ved en nøgleordsstyret metode indeholder regnearket nøgleord, der beskriver de handlinger der skal udføres, (også kaldet handlingsord) og testdata. Testere kan definere tests vha. nøgleordene uden at kende scriptsproget. Nøgleordene kan tilpasses til den applikation der testes.

Der er behov for teknisk ekspertise i scriptsproget til alle metoder (enten hos testere eller hos specialister i testautomatisering).

Uanset hvilken teknik der anvendes, skal man registrere de forventede resultater for hver test til senere sammenligning.

Statiske analyseværktøjer

Statiske analyseværktøjer, der anvendes på kildekode, kan håndhæve kodestandarder, men hvis de anvendes på eksisterende kode, kan de generere uoverskueligt mange meddelelser. Advarselsmeddelelser forhindrer ikke koden i at blive oversat til et eksekvérbart program, men bør ideelt set adresseres, så vedligeholdelse af koden bliver lettere i fremtiden. En gradvis implementering af analyseværktøjet med filtre indsat til udelukkelse af visse meddelelser vil være en effektiv metode at begynde.

Teststyringsværktøjer

Teststyringsværktøjer skal samarbejde med andre værktøjer eller regneark for at præsentere informationerne så de opfylder organisationens aktuelle behov.

6.3 Introduktion af et værktøj i en organisation (K1)	15 minutter

Termer

Ingen specifikke termer.

Baggrund

Hovedovervejelserne ved udvælgelse af et værktøj til en organisation omfatter:

- Vurdering af organisationens modenhed, styrker og svagheder og identifikation af muligheder for en forbedret testproces understøttet af værktøjer
- Vurdering i forhold til klare krav og formålskriterier
- En test af om den krævede funktionalitet er til stede og bestemmelse af, om produktet opfylder formålet (proof-of-concept)
- Vurdering af leverandøren (herunder undervisning, support og kommercielle aspekter)
- Identifikation af interne krav for coaching og mentoring i brug af værktøjet
- Vurdering af uddannelsesbehov i betragtning af testteamets nuværende testautomatiseringsfærdigheder
- Estimering af cost-benefit forholdet på basis af en konkret business case

Introduktion af det valgte værktøj i en organisation starter med et pilotprojekt, som har følgende formål:

- Lære flere detaljer om værktøjet
- Evaluere hvordan værktøjet passer til eksisterende processer og fremgangsmåder, og bestemme, hvad der skal ændres
- Beslutte standardmåder at anvende, styre, lagre og vedligeholde værktøjet og test-delprodukter (f.eks. fastsætte navnekonventioner for filer og tests, oprette biblioteker og definere modulopbygningen for testcases)
- Vurdere, om der kan opnås fordele for en fornuftig omkostning

Succesfaktorer for udbredelse af værktøjet i en organisation omfatter:

- Trinvis udrulning af værktøjet til resten af organisationen
- Tilpasning og forbedring af processerne, så de passer til anvendelse af værktøjet
- Tilbyde undervisning og coaching/mentoring af nye brugere
- Definere retningslinjer for anvendelse
- Implementere en måde at tage ved lære af anvendelsen af værktøjet
- Overvåge værktøjsanvendelse og fordele
- Yde support til testteamet for et specifikt værktøj
- Opsamling af opnåede erfaringer fra alle teams

Referencer

6.2.2 Buwalda, 2001, Fewster, 1999
6.3 Fewster, 1999

7. Referencer

Standarder

ISTQB Ordliste over termer anvendt i Software Testing Version 2.1

[CMMI] Chrissis, M.B., Konrad, M. og Shrum, S. (2004) CMMI, Guidelines for Process Integration and Product Improvement, Addison Wesley: Reading, MA
Se afsnit 2.1

[IEEE 829] IEEE Std 829™ (1998/2005) IEEE Standard for Software Test Documentation (er i øjeblikket under revision)

Se afsnittene 2.3, 2.4, 4.1, 5.2, 5.3, 5.5, 5.6
[IEEE 1028] IEEE Std 1028™ (1997) IEEE Standard for Software Reviews
Se afsnit 3.2

[IEEE 12207] IEEE 12207/ISO/IEC 12207-1996, Software life cycle processes
Se afsnit 2.1

[ISO 9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality
Se afsnit 2.3

Bøger

[Beizer, 1990] Beizer, B. (1990) Software Testing Techniques (2nd edition), Van Nostrand Reinhold: Boston
Se afsnittene 1.2, 1.3, 2.3, 4.2, 4.3, 4.4, 4.6

[Black, 2001] Black, R. (2001) Managing the Testing Process (3rd edition), John Wiley & Sons: New York
Se afsnittene 1.1, 1.2, 1.4, 1.5, 2.3, 2.4, 5.1, 5.2, 5.3, 5.5, 5.6

[Buwalda, 2001] Buwalda, H. et al. (2001) Integrated Test Design and Automation, Addison Wesley: Reading, MA
Se afsnit 6.2

[Copeland, 2004] Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: Norwood, MA
Se afsnittene 2.2, 2.3, 4.2, 4.3, 4.4, 4.6

[Craig, 2002] Craig, Rick D. og Jaskiel, Stefan P. (2002) Systematic Software Testing, Artech House: Norwood, MA
Se afsnittene 1.4.5, 2.1.3, 2.4, 4.1, 5.2.5, 5.3, 5.4

[Fewster, 1999] Fewster, M. og Graham, D. (1999) Software Test Automation, Addison Wesley: Reading, MA
Se afsnittene 6.2, 6.3

[Gilb, 1993]: Gilb, Tom og Graham, Dorothy (1993) Software Inspection, Addison Wesley: Reading, MA
Se afsnittene 3.2.2, 3.2.4
[Hetzel, 1988] Hetzel, W. (1988) Complete Guide to Software Testing, QED: Wellesley, MA
Se afsnittene 1.3, 1.4, 1.5, 2.1, 2.2, 2.3, 2.4, 4.1, 5.1, 5.3

[Kaner, 2002] Kaner, C., Bach, J. og Pettticord, B. (2002) Lessons Learned in Software Testing, John Wiley & Sons:
Se afsnittene 1.1, 4.5, 5.2

[Myers 1979] Myers, Glenford J. (1979) The Art of Software Testing, John Wiley & Sons:
Se afsnittene 1.2, 1.3, 2.2, 4.3

[van Veenendaal, 2004] van Veenendaal, E. (ed.) (2004) The Testing Practitioner (Kapitlerne 6, 8, 10),
UTN Publishers: The Netherlands
Se afsnittene 3.2, 3.3

8. Tillæg A – Pensum-baggrund

Historien om dette dokument

Dette dokument blev forberedt i perioden 2004 og 2011 af en arbejdsgruppe bestående af medlemmer udpeget af International Software Testing Qualifications Board (ISTQB). Det blev først reviewet af et udvalgt reviewpanel og derefter af repræsentanter fra det internationale softwaretestsamfund. De regler, der er anvendt ved fremstillingen af dette dokument, findes i Tillæg C.

Dokumentet er pensum for International Foundation Certificate in Software Testing, det første international kvalifikationsniveau godkendt af ISTQB (www.istqb.org).

Formål for Foundation Certificate qualification

- At opnå anerkendelse af test som en væsentlig og professionel softwareteknisk specialisering
- At levere en standardmodel til udvikling af testeres karriere
- At sørge for, at professionelt kvalificerede testere bliver anerkendt af arbejdsgivere, kunder og kolleger, og til at hæve testeres profil
- At fremme konsistent og god testpraksis inden for alle softwaretekniske discipliner
- At identificere testemner, der er relevante og værdifulde for industrien
- At sætte softwareleverandører i stand til at ansætte godkendte testere og dermed få en kommerciel fordel i forhold til deres konkurrenter ved at reklamere for deres ansættelsespolitik
- At skabe en mulighed for testere og de der har interesse i test til at opnå en internationalt anerkendt kvalifikation i emnet

Formål for international kvalifikation

(bearbejdet efter ISTQB-mødet i Sollentuna, November 2001)

- At blive i stand til at sammenligne testevner på tværs af forskellige lande
- At sætte testere i stand til lettere at bevæge sig på tværs af landegrænserne
- At gøre det muligt for multinationale eller internationale projekter at få en fælles forståelse af testemner
- At øge antallet af kvalificerede testere over hele verden
- At får større effekt/værdi som et internationalt baseret initiativ i stedet for en landespecifik metode
- At udvikle en fælles international enhed med forståelse og kendskab til test via dette pensum og terminologi, og til at øge kendskabsniveauet for test for alle deltagere
- At fremme test som et erhverv i flere lande
- At sætte testere i stand til at opnå en anerkendt kvalifikation på deres eget sprog
- At gøre det muligt at dele kendskab og ressourcer på tværs af lande
- At opnå international anerkendelse af testere og denne kvalifikation gennem deltagelse fra mange lande

Adgangskrav for denne kvalifikation

Adgangskriteriet for at tage ISTQB Foundation Certificate in Software Testing-eksamen er at kandidater har en interesse i softwaretest. Det anbefales imidlertid på det kraftigste, at kandidater også:

- Har lidt baggrund i enten softwareudvikling eller softwaretest, f.eks. seks måneders erfaring som system- eller brugeraccepttester eller som softwareudvikler
- Tager et kursus, der er godkendt til ISTQB-standarder (af et af de ISTQB-anerkendte nationale nævn)

Baggrund og historie for Foundation Certificate in Software Testing

Den uafhængige certificering af softwaretestere begyndte i Storbritannien gennem British Computer Societys Information Systems Examination Board (ISEB), da der blev etableret et softwaretestnævn i 1998 (www.bcs.org.uk/iseb). I 2002 begyndte ASQF i Tyskland at arbejde for en tysk kvalifikationsordning for testere (www.asqf.de). Dette pensum er baseret på ISEB- og ASQF-syllabierne; den omfatter reorganiseret, opdateret og lidt nyt indhold, og der er lagt målrettet vægt på emner, der vil give mest praktisk hjælp til testere.

Et eksisterende Foundation Certificate i Software Testing (f.eks. fra ISEB, ASQF eller et ISTQB-anerkendt nationalt nævn), der er godkendt før International Certificate blev frigivet, anses for at svare til International Certificate. Foundation Certificate bliver ikke forældet og behøver ikke at blive fornyet. Datoen, da det blev udstedt, er vist på certifikatet.

I hvert af de deltagende lande bliver lokale forhold kontrolleret af et nationalt ISTQB-anerkendt softwaretestnævn. De nationale nævns opgaver er specificeret af ISTQB, men er implementeret i hvert enkelt land. Opgaverne i landenævnene forventes at omfatte akkreditering af kursusudbydere og afholdelse af eksamen.

9. Tillæg B – Undervisningsmål og vidensniveau

Følgende undervisningsmål er defineret som gældende for dette pensum. Der bliver eksamineret i alle emne i pensum i henhold til deres undervisningsmål.

Niveau 1: Huske (K1)

Kandidaten kender, kan huske og genkalde sig en term eller et koncept.

Eksempel

Kan huske definitionen på en "afvigelse" som:

- "manglende levering af service til en slutbruger eller enhver anden interessent" eller
- "aktuel anderledes opførsel af komponenten eller systemet i forhold til dets forventede levering, service eller resultat"

Niveau 2: Forstå (K2)

Kandidaten kan udvælge årsager eller forklaringer for udsagn, der er relateret til emnet og kan opsummere, sammenligne, klassificere, kategorisere og give eksempler på testkonceptet.

Eksempler

Kan forklare årsagen til hvorfor tests bør designes så tidligt som muligt:

- For at finde defekter, når de er billigere at fjerne
- For at finde de vigtigste defekter først

Kan forklare ligheder og forskelle mellem integrations- og systemtest:

- Ligheder: test af mere end en komponent og kan teste ikke-funktionelle aspekter
- Forskelle: integrationstest koncentrerer sig om grænseflade og samspil, og systemtest koncentrerer sig om forhold for hele systemet, som f.eks. start-til-slut bearbejdning

Niveau 3: Anvende (K3)

Kandidaten kan udvælge den anvendelse for et koncept eller teknik og bruge det i en given kontekst.

Eksempel

- Kan identificere grænseværdier for gyldige og ugyldige partitioner
- Kan vælge testcases fra et givent tilstandsovergangsdiagram for at dække alle overgange

Niveau 4: Analysere (K4)

Kandidaten kan opdele oplysninger, der er relateret til en procedure eller en teknik, i deres grundbestanddele for at få en bedre forståelse og kan skelne mellem fakta og forstyrrelser. En typisk anvendelse er at foretage en analyse af et dokument, et stykke software eller en projektsituation og at foreslå passende handlinger til løsning af et problem eller en opgave.

Nøgleord: Analysere, organisere, finde sammenhæng, integrere, skitser, analysere, strukturere, egenskab, dekonstruere, skelne, forskelsbehandle, kende forskel, fokusere, vælge

Eksempel

- Analysere produktrisici og foreslå forebyggende og korrigerende mildnende aktiviteter
- Beskrive hvilke dele af en hændelsesrapport, der er faktiske, og hvilke der er afledt af resultater

Reference

(For kognitive niveauer af undervisningsmål)
Anderson, L. W. og Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon:

10. Tillæg C – Gældende regler for ISTQB

Foundation pensum

De regler, der er oplistet her, blev anvendt ved udvikling og review af dette pensum. (Der er vist et "TAG" efter hver regel som forkortelse for reglen.)

10.1.1 Generelle regler

SG1. Pensum bør være forståelig og til at forholde sig til for personer med nul til seks måneders (eller mere) erfaring i test. (6-MONTH)

SG2. Pensum bør være mere praktisk end teoretisk. (PRACTICAL)

SG3. Pensumt bør være klar og entydig for dens tiltænkte læsere. (CLEAR)

SG4. Pensum bør være forståelig for personer fra forskellige lande og være let at oversætte til forskellige sprog. (TRANSLATABLE)

SG5. Pensum bør bruge amerikansk engelsk. (AMERICAN-ENGLISH)

10.1.2 Aktuelt indhold

SC1. Pensum bør indeholde de nyeste testkoncepter og bør reflektere den aktuelle bedste praksis i softwaretest, hvor der er general enighed om det. Pensum skal reviewes hvert tredje til femte år. (RECENT)

SC2. Pensum bør minimere tidsspecifikke emner, som f.eks. aktuelle markedsbetingelser, så det kan have en levetid på tre til fem år. (SHELF-LIFE).

10.1.3 Undervisningsmål

LO1. Undervisningsmål bør skelne mellem emner, der skal kunne huskes (kognitivt niveau K1), emner, som kandidaten bør forstå begrebsmæssigt (K2), og de som kandidaten bør være i stand til at praktisere/anvende (K3). (KNOWLEDGE-LEVEL)

LO2. Beskrivelsen af indholdet bør være i overensstemmelse med undervisningsmålet. (LOCONSISTENT)

LO3. For at illustrere undervisningsmålet bør prøveeksamensspørgsmål for hvert hovedafsnit udgives sammen med pensum. (LO-EXAM)

10.1.4 Overordnet struktur

ST1. Pensums struktur bør være klar, og tillade krydsreferencer til og fra andre dele, fra eksamensspørgsmål og fra andre relevante dokumenter. (CROSS-REF)

ST2. Overlap mellem afsnittene i pensum bør minimeres. (OVERLAP)

ST3. Hvert afsnit i pensum bør have samme struktur. (STRUCTURE-CONSISTENT)

ST4. Pensum bør indeholde version, udstedelsesdato og sidenummer på hver side. (VERSION)

ST5. Pensum bør indeholde vejledning med angivelse af den tid, der skal anvendes på hvert afsnit (for at afspejle vigtigheden af hvert emne). (TIME-SPENT)

Referencer

SR1. Der opgives kilder og referencer for koncepter i pensum for at hjælpe kursusudbydere med at finde flere oplysninger om emnet. (REFS)

SR2. Hvor der ikke er nogen let identificerbare og klare kilder, bør der være opgivet flere detaljer i pensum. F.eks. er definitioner givet i ordlisten, så det kun er termerne, der er oplistet i pensum. (NON-REF DETAIL)

Informationskilder

De termer, der er anvendt i pensum, er defineret i ISTQB's ordliste over termer, der anvendes i softwaretest. Der findes en tilgængelig version af ordlisten fra ISTQB.

Der er også medtaget en liste over anbefalede bøger om softwaretest. Listen over de vigtigste bøger er en del af referenceafsnittet.

11. Tillæg D – Bemærkning til kursusudbydere

Hvert hovedemne i pensum er tildelt en tid i minutter. Formålet med dette er både at give en vejledning om den relative mængde tid, der skal afsættes til hvert afsnit i et godkendt kursus og for at give et overslag over det minimum af tid der skal bruges til undervisning for hvert afsnit. Undervisere må bruge mere tid end angivet, og kandidater må også bruge mere tid på læsning og studie. Et kursusforløb behøver ikke at følge samme rækkefølge som den, der er givet i pensum.

Pensum indeholder referencer til etablerede standarder, som skal anvendes ved forberedelsen af undervisningsmaterialet. Hver standard, der anvendes, skal være den der er angivet i den aktuelle version af dette pensum. Der må også anvendes og refereres til andre publikationer, skabeloner og standarder, der ikke er refereret til i dette pensum, men der vil ikke blive eksamineret i dem.

Alle K3 og K4 undervisningsmål kræver en praktisk øvelse, der skal indgå i undervisningsmaterialet.

12. Tillæg E – Releasenotat

2010 Udgivelsen

1. Ændringer til undervisningsmål (LO) inkluderer nogle præciseringer
   a. Ordlyd ændret i følgende LO'er (indhold og niveau af LO bevares uændret): LO-1.2.2, LO-1.3.1, LO-1.4.1, LO-1.5.1, LO-2.1.1, LO-2.1.3, LO-2.4.2, LO-4.1.3, LO-4.2.1, LO-4.2.2, LO-4.3.1, LO-4.3.2, LO-4.3.3, LO-4.4.1, LO-4.4.2, LO-4.4.3, LO-4.6.1, LO-5.1.2, LO-5.2.2, LO-5.3.2, LO-5.3.3, LO-5.5.2, LO-5.6.1, LO-6.1.1, LO-6.2.2, LO-6.3.2.
   b. LO-1.1.5 er blevet omformuleret og opgraderet til K2, idet en sammenligning af defekt-relaterede termer kan forventes.
   c. LO-1.2.3 (K2) er blevet tilføjet. Indholdet var allerede dækket i 2007-pensum.
   d. LO-3.1.3 (K2) kombinerer nu indholdet af LO-3.1.3 og LO-3.1.4.
   e. LO-3.1.4 er blevet fjernet fra 2010-pensum, da det delvist er overflødigt med LO-3.1.3.
   f. LO-3.2.1 er blevet omformuleret for overensstemmelse med indholdet af 2010-pensum.
   g. LO-3.3.2 er blevet ændret, og dets niveau er blevet ændret fra K1 til K2, for overensstemmelse med LO-3.1.2.
   h. LO 4.4.4 er blevet ændret for klarhed, og er blevet ændret fra en K3 til K4. Årsag: LO-4.4.4 var allerede skrevet i en K4 måde.
   i. LO-6.1.2 (K1) er fjernet fra 2010-pensum og erstattet med LO-6.1.3 (K2). Der er ingen LO-6.1.2 i 2010-pensum.
2. Konsekvent brug af testtilgang i henhold til definitionen i ordlisten. Termen teststrategi vil ikke være påkrævet som en term at huske.
3. Kapitel 1.4 indeholder nu konceptet sporbarhed mellem testgrundlag og testcases
4. Kapitel 2.x indeholder nu testobjekter og testgrundlag
5. Gentest er nu hovedtermen i ordlisten i stedet for bekræftelsestest
6. Aspektet datakvalitet og test er blevet tilføjet flere steder i pensum: datakvaliteten og risiko i kapitel 2.2, 5.5, 6.1.8
7. Kapitel 5.2.3 Startkriterier er tilføjet som et nyt underkapitel. Årsag: Sammenhæng til slutkriterier (-> slutkriterier tilføjet i LO-5.2.9)
8. Konsekvent brug af termerne teststrategi og testtilgang med deres definition i ordlisten
9. Kapitel 6.1 er blevet afkortet da værktøjsbeskrivelserne var for store til en 45 minutters lektion
10. IEEE Std 829:2008 er blevet frigivet. Denne version af pensum tager ikke denne nye udgave i betragtning. Afsnit 5.2 refererer til dokumentet Master Test Plan. Indholdet af Master Test Plan er omfattet af konceptet, at dokumentet "Testplan" dækker forskellige niveauer: Testplaner for de enkelte testniveauer kan oprettes ligesom en testplan på projektniveau kan oprettes dækkende for flere testniveauer. Sidstnævnte kaldes hovedtestplan i dette pensum og i ISTQB ordlisten.
11. Etisk kodeks er blevet flyttet fra CTAL til CTFL

2011 Udgivelsen

Ændringer, der foretages med "vedligeholdelseudgivelse" 2011

1. Generelt: Gruppen erstattet af arbejdsgruppen
2. Erstattede post-conditions med postconditions for at være i overensstemmelse med ISTQB Ordliste 2.1.

3. Første forekomst: ISTQB erstattet af ISTQB ®
4. Introduktion til pensum: Beskrivelser af vidensniveau er fjernet, da det var overflødigt pga. tillæg B.
5. Afsnit 1.6: Da det ikke var hensigten at definere et undervisningsmål for de ”Etiske regler", er videsniveauet for afsnittet blevet fjernet.
6. Afsnit 2.2.1, 2.2.2, 2.2.3 og 2.2.4, 3.2.3: Rettet formateringsproblem i lister.
7. Afsnit 2.2.2 Ordet afgivelse var ikke korrekt i "... isolere afvigelse til en bestemt komponent ...". Derfor erstattet med "defekt" i denne sætning.
8. Afsnit 2.3: Rettet formatering af punktliste over testmål i relation til testtermerne, i afsnit Testtyper (K2).
9. Afsnit 2.3.4: opdateret beskrivelse af debugging for at være i overensstemmelse med version 2.1 af ISTQB Ordliste.
10. Afsnit 2.4 ordet "omfattende" fjernes fra "indeholder omfattende regressionstest", idet "omfattende" afhænger af ændringen (størrelse, risici, værdi, osv.) som skrevet i den næste sætning.
11. Afsnit 3.2: Ordet "herunder" er blevet fjernet for at præcisere sætningen.
12. Afsnit 3.2.1: Da aktiviteterne i et formelt review var forkert formateret, havde reviewprocessen 12 hovedaktiviteter i stedet for seks, som er hensigten. Dette er ændret tilbage til seks, hvilket gør dette afsnit i overensstemmelse med 2007-pensummet og ISTQB Advanced Level pensum 2007.
13. Afsnit 4: Ordet "udviklet" erstattet med "defineret", idet testcases bliver defineret og ikke udviklet.
14. Afsnit 4.2: Tekst ændres for at præcisere, hvordan black-box og white-box test kan bruges sammen med erfaringsbaserede teknikker.
15. Afsnit 4.3.5 tekstændring ".. mellem aktører, herunder brugere og system .." til "... mellem aktører (brugere eller systemer), ...".
16. Afsnit 4.3.5 alternative stier erstattes af alternative scenarier.
17. Afsnit 4.4.2: For at præcisere ordet forgreningstest i teksten i afsnit 4.4, er en sætning blevet ændret for at præcisere fokus af forgreningstest.
18. Afsnit 4.5, afsnit 5.2.6: Udtrykket "experienced-based" er blevet erstattet af det rigtige udtryk " experience-based " (engelsk udgave af pensum).
19. Afsnit 6.1: overskrift "6.1.1 Forstå meningen og formålet med værktøjsstøtte til test (K2)" ændret til "6.1.1 Værktøjstøtte til test (K2)".
20. Afsnit 7 / Bøger: Den 3. udgave af [Black, 2001], erstatter 2. udgave.
21. Tillæg D: kapitler som kræver øvelser er blevet erstattet af det generelle krav om, at alle undervisningsmål K3 og højere kræver øvelser. Dette er et krav angivet i ISTQB akkrediteringsprocessen (version 1.26).
22. Bilag E: De ændrede undervisningsmål mellem version 2007 og 2010 er nu korrekt angivet.

13. Indeks