

Sertifisert Tester

Pensum for grunn-nivå ("Foundation Level")

Engelsk Versjon 2010 - Norsk Versjon 2010.N1

28.09.2010

Norwegian Testing Board
International Software Testing Qualifications Board



Copyright © 2010 forfatterne av den norske oversettelsen (Hans Schaefer, Skule Johansen, Jürgen Richter, Thomas Borchsenius, Ragnhild Egtvedt, Berit Kristine Hatten, Kjersti Forthun, Ernst von Düring).

Copyright © 2008 forfatterne av den norske oversettelsen (Hans Schaefer, Monika Stöcklein-Olsen, Skule Johansen, Kjersti Forthun, Lillann Solberg, Jürgen Richter).

Copyright © 2010 forfatterne for oppdateringen 2010 (Thomas Müller (formann))

Copyright © 2007 forfatterne for oppdateringen 2007 (Thomas Müller (formann), Dorothy Graham, Debra Friedenberga og Erik van Veendendal)

Copyright © 2005 forfatterne (Thomas Müller (formann), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson og Erik van Veendendal)

Alle rettigheter forbeholdt.

Forfatterne overfører sin copyright til International Software Testing Qualifications Board (ISTQB). Forfatterne (som nå har copyright) og ISTQB (som i framtiden har copyright) er enige om følgende betingelser for bruk av dette dokument:

- 1) Enhver person eller opplæringsfirma kan bruke dette pensum som grunnlag for kurs, hvis forfatterne og ISTQB blir anerkjent som kilde og eiere av copyright av dette pensum (syllabus) og under forutsetning av at hver annonsering eller markedsføring av et slikt kurs kan nevne dette pensum bare etter at kursmaterialet er levert til offisiell akkreditering til et av ISTQB anerkjent nasjonalt Board.
- 2) Enhver person eller gruppe av personer kan bruke dette pensum som basis for artikler, bøker eller annen type skrevet materiale hvis forfatterne og ISTQB blir anerkjent og nevnt som kilde og eiere av copyright av dette dokument.
- 3) Alle ISTQB-ankjente nasjonale Board kan oversette dette pensum og gi lisens på dette materiale (eller dets oversettelse) videre til andre grupper.

Revisjonshistorie

Versjon	Dato	Bemerkninger
2010.N1	17.12.2010	Full gjennomgang av HS mot offisiell 2010-versjon på engelsk. Publisert 20.12.2010 av NTB.
NTB	30.9.2010	Norsk versjon ferdig for review
NTB	15-10-2008	Mindre språklige rettelser
Norwegian Testing Board 2008	03-Sept-2008	Første norske versjon
ISTQB 2007 - engelsk	01-May-2007	Certified Tester Foundation Level Syllabus Maintenance Release Earlier English versions mentioned in that document.

Innholdsfortegnelse

Innledning	8
1.1 <i>Formål med dette dokument</i>	8
1.2 <i>Sertifisert tester på grunnnivå (Foundation Level) i programvaretesting</i>	8
1.3 <i>Læremål / kunnskapsnivå</i>	8
1.4 <i>Eksamen</i>	8
1.5 <i>Akkreditering</i>	8
1.6 <i>Detaljnivå</i>	9
1.7 <i>Hvordan dette pensum er organisert</i>	9
1.8 <i>Forklaring av spesiell terminologibruk på norsk</i>	9
1 Grunnlag for testingen (K2).....	10
1.1 Hvorfor er test nødvendig (K2).....	11
1.1.1 Programvaresystemer i en større sammenheng (K1)	11
1.1.2 Grunner til programvarefeil (K2)	11
1.1.3 Testings rolle ved programvareutvikling, vedlikehold og drift (K2)	11
1.1.4 Testing og kvalitet (K2)	11
1.1.5 Hvor mye testing er nok? (K2)	12
1.2 Hva er testing? (K2)	13
1.3 Syv testprinsipper (K2).....	14
1.4 Den grunnleggende testprosessen (K1)	15
1.4.1 Testplanlegging og -styring (K1).....	15
1.4.2 Testanalyse og -design (K1).....	15
1.4.3 Testimplementering og -utførelse (K1)	16
1.4.4 Evaluering av sluttkriterier og rapportering (K1)	16
1.4.5 Avslutningsaktiviteter (K1)	16
1.5 Testings psykologi (K2).....	18
1.6 Etikk for testeren (K2).....	19
2 Testing gjennom livssyklusen (K2)	20
2.1 Utviklingsmodeller for programvare(K2).....	21
2.1.1 V-modellen (sekvensiell utviklingsmodell) (K2)	21
2.1.2 Iterativ-inkrementell utviklingsmodeller (K2)	21
2.1.3 Testing i en livssyklusmodell (K2).....	21
2.2 <i>Testnivåer</i> (K2).....	23
2.2.1 Komponenttesting (K2)	23
2.2.2 Integrasjonstesting (K2)	24
2.2.3 Systemtesting (K2).....	25
2.2.4 Akseptansetesting (K2).....	25
2.3 <i>Testtyper</i> (K2).....	28
2.3.1 Testing av funksjonalitet (funksjonell testing) (K2)	28
2.3.2 Testing av ikke-funksjonelle egenskaper av programvaren (ikke-funksjonell testing) (K2).....	28
2.3.3 Testing av programvarens struktur/arkitektur (strukturell testing, hvit boks testing) (K2).....	29
2.3.4 Testing relatert til endringer (re-testing og regresjonstesting) (K2)	29
2.4 <i>Testing i vedlikehold</i> (K2).....	30
3 Statiske teknikker (K2)	31
3.1 <i>Statiske teknikker og testprosessen</i> (K2).....	32
3.2 <i>Granskningsprosessen</i> (K2).....	33
3.2.1 Faser i en formell granskning (K1).....	33
3.2.2 Roller og ansvar (K1)	34
3.2.3 Granskningstyper (K2)	34
3.2.4 Suksessfaktorer for granskninger (K2)	35
3.3 <i>Statisk analyse med verktøy</i> (K2)	36
4 Teknikker for testdesign (K3)	37
4.1 Testutviklingsprosessen (K3)	39
4.2 Kategorier for teknikker for testdesign (K2)	40
4.3 Spesifikasjonsbaserte (svart boks) teknikker (K3)	41

4.3.1	Ekvivalensklasseinndeling (K3)	41
4.3.2	Grenseverdianalyse (K3)	41
4.3.3	Beslutningstabelltesting (K3)	41
4.3.4	Tilstandsbasert testing (K3)	42
4.3.5	Brukstilfelle (use case) testing (K2)	42
4.4	Strukturbaserte (hvit boks) teknikker (K4)	43
4.4.1	Programinstruksjonstesting og dekning (statement testing and coverage) (K3)	43
4.4.2	Forgrenings- / Beslutningstesting og dekning (branch or decision testing and coverage) (K3)	43
4.4.3	Andre strukturbaserte teknikker (K1)	43
4.5	Erfaringsbaserte teknikker (K2)	45
4.6	Valg av testteknikker (K2)	46
5	Testledelse (K3)	47
5.1	Testorganisasjon (K2)	49
5.1.1	Testorganisasjon og uavhengighet (K2)	49
5.1.2	Testerens og testlederens oppgaver (K1)	49
5.2	Test planlegging og estimering (K3)	51
5.2.1	Testplanlegging (K2)	51
5.2.2	Testplanleggingsaktiviteter (K2)	51
5.2.3	Startkriterier (K2)	51
5.2.4	Sluttkriterier (K2)	52
5.2.5	Testestimering (K2)	52
5.2.6	Teststrategier og testmetoder (K2)	52
5.3	Overvåking og kontroll av testframdriften (K2)	54
5.3.1	Oppfølging av testframdrift (K1)	54
5.3.2	Testrapportering (K2)	54
5.3.3	Teststyring (K2)	54
5.4	Konfigurasjonsstyring (K2)	56
5.5	Risiko og testing (K2)	57
5.5.1	Prosjektrisikoeer (K2)	57
5.5.2	Produktrisikoeer (K2)	57
5.6	Hendelses- og feilhåndtering (K3)	59
6	Verktøystøtte til test (K2)	61
6.1	Typer testverktøy (K2)	62
6.1.1	Forstå mål og mening med verktøystøtte for testing (K2)	62
6.1.2	Klassifisering av testverktøy (K2)	62
6.1.3	Verktøystøtte for administrasjon av testing og tester (K1)	63
6.1.4	Verktøystøtte for statisk testing (K1)	63
6.1.5	Verktøystøtte for testspesifikasjon (K1)	64
6.1.6	Verktøystøtte for testutføring og logging (K1)	64
6.1.7	Verktøystøtte til ytelse og overvåking (K1)	64
6.1.8	Verktøystøtte for spesifikke testbehov (K1)	65
6.2	Effektiv bruk av verktøy: mulige fordeler og risikoer (K2)	66
6.2.1	Mulige fordeler og risikoer ved verktøystøtte til testing (for alle verktøy) (K2)	66
6.2.2	Spesielle hensyn for noen typer verktøy (K1)	66
6.3	Introduksjon av et verktøy i en organisasjon (K1)	68
7	Referanser	69
7.1	Standarder	69
7.2	Bøker	69
8	Vedlegg A – Bakgrunn til pensum	71
8.1	Historien til dette dokumentet	71
8.2	Mål med kvalifikasjonen med "Foundation Certificate"	71
8.3	Mål med den internasjonale kvalifikasjonen (tatt fra ISTQB møte på Sollentuna, november 2001)	71
8.4	Startkrav for denne kvalifikasjonen	71

8.5	<i>Bakgrunn og historie for grunnsertifikatet (Foundation Certificate in Software Testing)</i>	72
9	Vedlegg B – Læremål / kunnskapsnivå	73
9.1	<i>Nivå 1: Husk (K1)</i>	73
9.2	<i>Nivå 2: Forstå (K2)</i>	73
9.3	<i>Nivå 3: Bruke (K3)</i>	73
9.4	<i>Nivå 4: Analysere (K4)</i>	73
10	Vedlegg C – Regler som er brukt av ISTQB	75
	<i>Grunnivå-pensum (Foundation syllabus)</i>	75
	Generelle regler	75
	Aktuelt innhold	75
	Læremål	75
	Totalstruktur	75
11	Vedlegg D – Informasjon til kursholdere	77
12	Appendix E – Release Notes Syllabus 2010	78

Takk

International Software Testing Qualifications Board - arbeidsgruppe Foundation Level (utgave 2010): Thomas Müller (formann)

International Software Testing Qualifications Board - arbeidsgruppe Foundation Level (utgave 2007): Thomas Müller (formann), Dorothy Graham, Debra Friedenber og Erik van Veendendal.

Forfatterne takker de som har gjennomgått den engelske originalteksten (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson og Wonil Kwon) og alle nasjonale Boards for deres forslag til den nåværende versjonen av pensum.

International Software Testing Qualifications Board - arbeidsgruppe Foundation Level (utgave 2005): Thomas Müller (formann), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson og Erik van Veendendal. Forfatterne takker de som har gjennomgått teksten og alle nasjonale Boards for deres forslag til det aktuelle pensum.

Oversettelsen er utført av medlemmer i Norwegian Testing Board. I tillegg har Leo Eliassen levert verdifulle kommentarer. Disse er spesialister i softwaretesting, men ikke profesjonelle oversettere. Oversettelsen er gjort på dugnad.

Oversettelsen holder seg tett til den engelske originalen, fordi denne blir brukt i den internasjonale sertifiseringen, og vi vil sikre at oversettelsen dekker alle aspekter og nyanser ved originalen. Vi er klar over at språket derfor delvis virker noe tungt.

Vi er takknemlige for all tilbakemelding. Vi skal også besvare spørsmål du måtte ha innen rimelig tid. Kommentarer til Hans Schaefer (hans.schaefer@ieee.org).

Innledning

1.1 Formål med dette dokument

Dette pensum er grunnlaget for International Software Testing Qualification på grunnivå (Foundation Level). International Software Testing Qualifications Board (ISTQB) gir det til de nasjonale Boards for å akkreditere kursholdere og for å lage eksamensspørsmål på deres nasjonale språk. Kursholderne lager kursmateriale og bestemmer passende læringsmetoder for akkreditering, og dette pensum vil hjelpe kandidater i deres forberedelse til eksamen. Informasjon om historie og bakgrunn for dette pensum finnes i vedlegg A.

1.2 Sertifisert tester på grunnivå (Foundation Level) i programvaretesting

Målgruppen for sertifiseringen på grunnivå (Foundation Level) er enhver som er involvert i test av programvare. Dette omfatter personer i roller som testere, testutviklere, testingeniører, testkonsulenter, testledere, brukere som akseptansetestere og programvareutviklere. Grunnivåsertifiseringen er også passende for den som vil ha en grunnleggende forståelse av test av programvare, som for eksempel prosjektledere, kvalitetsledere, utviklingsledere for programvare, forretningsanalytikere, IT-direktører og ledelseskonsulenter. De som innehar dette sertifikat kan så gå videre til en kvalifikasjon på test av programvare på et høyere nivå.

1.3 Læremål / kunnskapsnivå

Kunnskapsnivå blir gitt for hvert avsnitt i dette dokument og klassifiseres som følgende::

- K1: huske, gjenkjenne, kunne gjengi
- K2: forstå, forklare, gi begrunnelser, sammenligne, klassifisere, kategorisere, gi eksempler, oppsummere
- K3: anvende, bruke
- K4: analysere

Mer detaljer og eksempler av læremål er gitt i vedlegg B.

Alle uttrykk under overskriften "Begreper" rett under hver kapitteloverskrift skal huskes (K1), selv om de ikke blir uttrykkelig nevnt i læremålene. De er forklart i ISTQB Glossary.

1.4 Eksamen

Eksamen på grunnivå vil bli basert på dette pensum. Svar til eksamensspørsmål kan kreve bruk av materiale fra mer enn ett avsnitt i dette pensum. Alle deler av pensum kan brukes til eksamen.

Eksamensformat er flervalg (multiple choice).

Eksamen kan tas som del av et akkreditert kurs eller uavhengig av kurs etter avtale med Norwegian Testing Board. Det stilles ikke krav til gjennomført akkreditert kurs for å stille til eksamen.

1.5 Akkreditering

Kursholdere og deres kursmateriale som følger dette pensum, kan akkrediteres av et nasjonalt Board anerkjent av ISTQB. Retningslinjer for akkreditering bør hentes fra det nasjonale Board eller det Board som foretar akkreditering. Et akkreditert kurs som er anerkjent i samsvar med dette pensum og som holdes av akkreditert instruktør, har tillatelse til å få arrangert en ISTQB-eksamen som del av kurset.

Videre veiledning til kursholdere er gitt i vedlegg D.

1.6 Detaljnivå

Detaljnivået i dette pensum tillater internasjonalt sammenlignbar læring og prøver. For å få dette til, består det av:

- Generelle læremål som beskriver formålet med grunnivået (Foundation Level).
- En liste med informasjon som skal læres bort, inklusive en beskrivelse og referanser til ekstra kilder hvis nødvendig.
- Læremål for hvert kunnskapsområde, som beskriver hva folk skal kunne og forstå.
- En liste med begreper som kandidatene må kunne gjenta og ha forstått.
- En beskrivelse av nøkkelkonsepter som skal læres, inklusive kilder som alminnelig akseptert litteratur og standarder.

Pensums innhold er ikke en beskrivelse av hele kunnskapsområdet til testing av programvare. Det beskriver bare detaljnivået som skal dekkes i grunnleggende kurs.

1.7 Hvordan dette pensum er organisert

Det er seks hovedkapitler. Hovedoverskriften viser høyeste nivå av læremålene som er dekket i kapitlet og spesifiserer kurstiden for kapitlet. For eksempel:

2. Testing gjennom livssyklusen (K2)	115 minutter
--------------------------------------	--------------

Denne overskriften viser at kapittel 2 har læremål på nivå K1 (implisert når et høyere nivå er vist) og K2 (men ikke K3), og burde ta 115 minutter for å lære bort materiellet i kapitlet. Innenfor hvert kapittel er et antall avsnitt. Hvert avsnitt har også læremål og en tidsmengde som er påkrevd. Underavsnitt har ikke noe tid allokert og er innbefattet i tiden for avsnittet.

1.8 Forklaring av spesiell terminologibruk på norsk

I en del situasjoner er det vanskelig å finne treffende begreper på norsk, eller det er brukt alternative formuleringer. I dette avsnitt er slike språklige problemer forklart

Test vs. Testing: De to ordene er brukt med samme betydning i pensum.

Feil: På engelsk finnes begrepene error, mistake, fault, defect, problem, failure. På norsk bruker vi samme ord for alle disse begrepene, nemlig feil. For å kunne skille mellom årsaken til en feil, selve feilen og de følgene en feil har når den blir utført, brukes de engelske uttrykk i den norske eksamen.

White box testing: Her har vi valgt å bruke de engelske begrepene på ulike testmetoder og dekningsgrader sammen med de norske.

Review og granskning: Begge ord er bruk i samme betydning.

1 Grunnlag for testingen (K2)	155 minutter
--------------------------------------	---------------------

Læremål

Målene identifiserer hva du skal kunne etter fullførelsen av hver modul.

1.1 Hvorfor er test nødvendig (K2)

- LO-1.1.1 Beskriv med eksempler hvordan en feil kan skade en person, miljøet eller en bedrift. (K2)
- LO-1.1.2 Skill mellom årsaken til en feil og feilens følger. (K2)
- LO-1.1.3 Begrunn hvorfor det er nødvendig å utføre test ved å gi eksempler. (K2)
- LO-1.1.4 Beskriv hvorfor test er en del av kvalitetssikringen og gi eksempler på hvordan test hjelper med å øke kvaliteten. (K2)
- LO-1.1.5 Forklar og sammenlign begrepene feil (engelsk: error, mistake, bug, defect, failure, fault). (K2)

1.2 Hva er testing (K2)

- LO-1.2.1 Husk de vanlige formål for test. (K1)
- LO-1.2.2 Gi eksempler for målsetningen for testing i ulike faser i livssyklusmodellen for programvare. (K2)
- LO-1.2.3 Skill mellom testing og debugging (K2)

1.3 Syv testprinsipper (K2)

- LO-1.3.1 Forklar de syv prinsippene ved testing. (K2)

1.4 Den grunnleggende testprosessen (K1)

- LO-1.4.1 Husk de fem grunnleggende testaktivitetene og tilhørende oppgaver fra planlegging til avslutning. (K1)

1.5 Testingens psykologi (K2)

- LO-1.5.1 Husk de psykologiske faktorene som har innflytelse på testsuksess. (K1)
- LO-1.5.2 Still testerens og utviklerens holdninger opp mot hverandre. (K2)

1.1 Hvorfor er test nødvendig (K2)	20 minutter
------------------------------------	--------------------

Begreper

Feil, systemfeil, feiltakelse, kvalitet, risiko
De engelske begrepene bug, defect, error, failure, fault, mistake.

1.1.1 Programvaresystemer i en større sammenheng (K1)

Programvaresystemer er en vanlig del av hverdagen, fra forretningsløsninger (f. eks. i bankbransjen) til forbrukerprodukter (f. eks. biler). Folk flest har opplevd at programvare ikke fungerer som forventet. Programvare som ikke fungerer korrekt kan føre til mange problemer, inkludert tap av penger, tid eller forretningsomdømme, og kan til og med medføre menneskelig skade eller død.

1.1.2 Grunner til programvarefeil (K2)

Et menneske kan gjøre en feil (mistake), som fører til en feil (defect, fault eller bug) i programkoden, eller i et dokument. Hvis en feil i koden innføres, vil systemet ikke gjøre hva det skal (eller det kan gjøre noe det ikke burde), noe som kan føre til feil (failure). Feil i programvare, systemer eller dokumenter kan føre til problemer (failures), men ikke alle feil gjør dette.

Feil inntreffer fordi mennesker generelt begår feil og på grunn av tidspress, kompleks kode, sammensatt infrastruktur, teknologiendringer, og/eller mye interaksjon mellom flere systemer.

Systemfeil kan også forårsakes av miljøfaktorer: For eksempel stråling, magnetisme, elektroniske felt og forurensning kan forårsake feil i firmware eller påvirke programvaren ved å forandre maskinvareforholdene.

1.1.3 Testings rolle ved programvareutvikling, vedlikehold og drift (K2)

Grundig testing av systemer og tilhørende dokumentasjon kan redusere risikoen for at problemer inntreffer i drift. Dette kan også bidra til å høyne kvaliteten av et programvaresystem, men forutsetningen er at de feil som er funnet blir rettet før systemet blir frigitt til drift.

Programvaretesting er også ofte påkrevd ifølge kontrakter, eller andre juridiske krav, eller industrispesifikke standarder.

1.1.4 Testing og kvalitet (K2)

Ved hjelp av testing er det mulig å måle programvarekvaliteten ved å referere til feil som er funnet, for både funksjonelle og ikke-funksjonelle programvarekrav og -egenskaper (f. eks. pålitelighet, brukervennlighet, effektivitet, vedlikeholdbarhet og portabilitet). For ytterligere informasjon om ikke-funksjonell testing, se kapittel 2; for mer informasjon om programvareegenskaper, se 'Software Engineering – Software Product Quality' (ISO 9126-1).

Testing kan gi tillit til programvarekvaliteten hvis få eller ingen feil blir påvist. En velkonstruert test som systemet takler kan redusere risikoen i et system. Hvis en test påviser feil, blir systemets kvalitet forbedret når disse feilene blir rettet.

Erfaring bør opparbeides fra tidligere prosjekter. Ved å forstå årsaken til feil som er funnet i andre prosjekter, kan prosesser forbedres, noe som burde forhindre at slike feil forekommer på nytt. På denne måten får man en forbedring av kvaliteten til fremtidige systemer. Dette er en del av kvalitetssikringen.

Testing bør integreres som en av kvalitetssikringsaktivitetene (dvs. ved siden av utviklingsstandarder, opplæring og feilanalyse).

1.1.5 Hvor mye testing er nok? (K2)

Beslutningsprosessen om hvor mye testing som bør foretas bør ta hensyn til risikonivå, inkludert teknisk, sikkerhets- og forretningsrelatert risiko, og prosjektbegrensinger slik som tid og budsjett. (Risiko er ytterligere behandlet i kapittel 5).

Testing bør gi tilstrekkelig informasjon til involverte parter slik at de kan foreta informerte beslutninger om frigivelsen av programvaren som testes, for det neste utviklingstrinnet eller for overlevering til kunder.

1.2 Hva er testing? (K2)

30 minutter

Begreper

Debugging, krav, granskning (review), testtilfelle, testing, testmål.

Bakgrunn

En vanlig forståelse av testing er at det bare dreier seg om kjøring av tester, dvs. utføring av programvaren. Dette er en del av testingen, men det finnes flere testaktiviteter.

Testaktiviteter eksisterer før og etter selve utførelsen av testen. Disse aktiviteter omfatter planlegging og styring, valg av testbetingelser, konstruksjon og utføring av testtilfelle, kontroll av resultater, evaluering av sluttkriterier, rapportering om testprosessen og systemet under test, og ferdigstilling eller avslutning etter at en testfase er gjennomført. Testing omfatter også granskning av dokumenter (inkludert kildekode) og å utføre statistisk analyse.

Både dynamisk og statistisk testing kan benyttes for å oppnå liknende formål, og vil gi informasjon som kan brukes til å forbedre både systemet som testes og utviklings- og testprosessene.

Det finnes forskjellige testformål:

- finne feil
- få tillit til kvalitetsnivået
- gi informasjon for å skaffe seg et beslutningsgrunnlag
- forebygge feil.

Tankeprosessen og aktivitetene som utføres ved å konstruere tester tidlig i livssyklusen (verifikasjon av testgrunnlaget via testdesign) kan hjelpe til å forhindre at feil blir introdusert i koden. Granskning av dokumenter (f.eks. krav) samt identifisering og løsning av problemer hjelper også til å forhindre feil i koden.

Ulike perspektiv i testingen reflekterer ulike formål. I utviklingstesting (f. eks. komponent-, integrasjons- og systemtester), kan hovedmålet være å forårsake så mange systemfeil som mulig slik at feil i systemet kan identifiseres og rettes. I akseptansetesting er hovedformålet som oftest å bekrefte at systemet fungerer som forventet, og få tillit til at systemet har innfridd kravene. I noen tilfeller er hovedformålet ved testingen å vurdere programvarekvaliteten (uten hensikt å rette eventuelle feil), for å gi informasjon til de involverte parter angående eventuell risiko som er knyttet til frigivelse av systemet til en gitt tidspunkt. Vedlikeholdstesting inkluderer ofte testing av at ingen nye feil har blitt introdusert mens endringer har blitt foretatt. Ved driftstesting er hovedformålet å evaluere systemets egenskaper, slik som pålitelighet og tilgjengelighet.

Debugging og testing er to forskjellige ting. Dynamisk testing kan identifisere feil (eng.: failures). Debugging er den utviklingsaktiviteten som finner og analyserer feilen og fjerner årsaken til feilen (failure) (reparerer den). Re-testing sjekker at feilen faktisk er korrigert. Ansvaret for disse aktivitetene er vanligvis forskjellig plassert, dvs. oftest er det slik: testere tester og utviklere debugger.

Testprosessen og dens aktiviteter er forklart i avsnitt 1.4.

1.3 Syv testprinsipper (K2)	35 minutter
-----------------------------	-------------

Begreper

Fullstendig testing.

Prinsipper

Et antall testprinsipper har blitt foreslått de siste 40 årene; disse gir generelle retningslinjer som gjelder for alle typer testing.

Prinsipp 1 – Test viser tilstedeværelsen av feil

Testing kan påvise feil, men kan ikke bevise fravær av feil. Testing reduserer sannsynligheten for at programvaren inneholder skjulte feil, men selv om ingen feil kan identifiseres er ikke det et bevis for korrekthet.

Prinsipp 2 – Fullstendig test er umulig

Fullstendig test (alle kombinasjoner av inndata og forbetninger) er ikke mulig unntatt i trivielle tilfeller. I stedet for fullstendig (uttømmende) test, bør risikoanalyse og prioritering benyttes for å fokusere testingen.

Prinsipp 3 – Tidlig testing

For å finne feil tidlig, bør testaktivitetene starte så tidlig som mulig i programvarens eller systemets utviklingslivssyklus, og de bør fokuseres på veldefinerte mål.

Prinsipp 4 – Klyngedannelse av feil

Testarbeidet bør bli fokusert proporsjonalt til forventet og senere observert feiltetthet i moduler. Et mindre antall moduler inneholder vanligvis mesteparten av feilene som blir funnet i testen før frigivelsen, eller er ansvarlige for de fleste feilene i drift.

Prinsipp 5 – "Pesticid-paradoks"

Hvis de samme testene gjentas om og om igjen, vil de etter hvert ikke finne nye feil. For å forhindre en slik situasjon, må testtilfellene regelmessig gjennomgås og revideres, og nye, forskjellige tester må utarbeides for å teste forskjellige deler av programvaren eller systemet for å finne mulige flere feil.

Prinsipp 6 – Test er avhengig av konteksten

Testing utføres på ulike måter i ulike kontekster. For eksempel testes sikkerhetskritisk programvare på en annen måte enn netthandelsprogramvare.

Prinsipp 7 – Feiltakelse vedr. "Fravær av feil"

Det hjelper lite å identifisere og rette feil hvis systemet er ubrukelig og ikke oppfyller brukernes behov og forventninger.

1.4 Den grunnleggende testprosessen (K1)	<i>35 minutter</i>
--	--------------------

Begreper

Re-testing, sluttkriterier, hendelse, regresjonstesting, testgrunnlag, testbetingelse, testdekning, testdata, testutførelse, testlogg, testplan, testprosedyre, testpolicy, testsuite, testsluttrapport, testware.

Bakgrunn

Den mest synlige delen av testprosessen er utførelsen av tester. For at en testplan skal være både effektiv og virkningsfull, bør imidlertid testplaner også ta med tid som skal brukes til planlegging av testen, design av testtilfelle, forberedelse av utførelsen og evaluering av resultater.

Den grunnleggende testprosessen består av følgende hovedaktiviteter:

- Planlegging og styring
- Analyse og design
- Implementering og utførelse
- Evaluering av sluttkriterier og rapportering
- Avslutningsaktiviteter

Selv om denne listen følger logisk rekkefølge, er det ofte slik at aktiviteter overlapper eller finner sted samtidig. Vanligvis kreves det tilpasning av disse hovedaktivitetene til systemet og/eller prosjektet.

1.4.1 Testplanlegging og -styring (K1)

Testplanlegging er den aktiviteten som består av å definere testens formål og spesifisere testaktivitetene for å oppfylle formål og oppdrag.

Teststyring er den kontinuerlige sammenligningen av virkelig fremdrift ift. planen, samt rapportering av status og avvik ift. planen. Det innebærer også å gjennomføre tiltak for å oppfylle prosjektets oppdrag og formål. For å kunne styre testingen, bør hele testaktivitetene overvåkes nøye gjennom hele prosjektet. Testplanlegging tar med i betraktning tilbakemelding fra overvåkings- og styringsaktiviteter.

Testplanleggings- og styringsaktiviteter er definert i kapittel 5 i dette pensum.

1.4.2 Testanalyse og -design (K1)

Testanalyse og -design er aktiviteten der de generelle testformålene blir omformet til konkrete testbetingelser og testtilfelle.

Testanalyse og -design har følgende hovedoppgaver:

- Granskning av testgrunnlaget (slik som krav, kritikalitetsnivå (risikonivå), rapporter fra risikoanalyse, arkitektur, design, grensesnittspesifikasjoner).
- Evaluering av testbarheten av testgrunnlaget og testobjektene.
- Identifisering og prioritering av testbetingelsene basert på en analyse av testobjektene, spesifikasjonen, atferd og struktur av programvaren.
- Utarbeidelse og prioritering av høynivå testtilfelle.
- Identifisering av nødvendige testdata for å støtte testbetingelsene og testtilfellene.
- Design av testmiljøet og identifisering av nødvendig infrastruktur og verktøy.
- Konstruksjon av sporbarhet i begge retninger mellom testbasis og testtilfelle.

1.4.3 Testimplementering og -utførelse (K1)

Testimplementering og -utførelse er aktiviteten der testprosedyrer og scripts spesifiseres ved å kombinere testtilfelle i særskilt orden og inkludere all annen informasjon som behøves for utførelsen av testen, og der testmiljøet settes opp og testene utføres.

Testimplementering og -utførelse har følgende hovedoppgaver:

- Ferdigstilling, implementering og prioritering av testtilfelle (inklusive identifikasjon av testdata).
- Utvikling og prioritering av testprosedyrer, utarbeidelse av testdata, og hvis det er behov, forberedelse av testrammer og skriving av automatiske testscripts.
- Produksjon av testsuiter fra testprosedyrene for å muliggjøre effektiv testutførelse.
- Verifisering av at testmiljøet har blitt satt opp på korrekt måte.
- Utførelse av testprosedyrer, enten manuelt eller ved bruk av testverktøy, ifølge en planlagt sekvens
- Logging av resultater fra testutføringen og registrering av identiteter og versjoner av programvaren under test, testverktøy og testware.
- Sammenligning av faktiske resultater med forventede resultater.
- Rapportering av avvik som hendelser og analyse av disse for å identifisere årsak (f. eks. en feil i koden, i spesifiserte testdata, i et testdokument, eller en feiltakelse i måten testen ble utført).
- Gjentakelse av testaktiviteter som følge av tiltak som ble utført i forbindelse med avvik. For eksempel, gjennomføring av en test som tidligere ikke lyktes for å bekrefte en feilrettelse (re-testing), utførelse av en korrigeret test og/eller utførelse av tester for å kontrollere at feil ikke har blitt introdusert i uendrede områder av programvaren eller at en feilretting ikke avslørte andre feil (regresjonstesting).

1.4.4 Evaluering av sluttkriterier og rapportering (K1)

Evaluering av sluttkriterier er den aktiviteten der testutførelsen blir vurdert i forhold til de definerte testformålene. Dette bør gjøres for hvert testnivå (se kapittel 2.2).

Evaluering av avslutningskriterier og rapportering har følgende hovedoppgaver:

- Kontroll av testlogger mot sluttkriteriene som ble spesifisert i testplanleggingen.
- Vurdering om flere tester behøves eller om sluttkriteriene bør forandres.
- Utarbeidelse av testsluttrapporten for de involverte parter.

1.4.5 Avslutningsaktiviteter (K1)

Her samles det inn data fra avsluttede testaktiviteter for å samle erfaring, testware, fakta og tall. Slike aktiviteter gjøres ved prosjektmilepeler for eksempel når et programvaresystem frigis, når et testprosjekt er ferdig (eller avbrytes), når en milepæl har blitt oppnådd, eller når en vedlikeholdsfrigivelse har blitt fullført.

Testingsavslutningsaktiviteter omfatter følgende hovedoppgaver:

- Kontroll av hvilke av de planlagte oppgavene er utført og testartefaktene som har blitt levert.
- Lukking av hendelsesrapporter eller opprettelse av endringsregistreringer for de som fortsatt er åpne.
- Dokumentasjon på at systemet er blitt godkjent eller underkjent.
- Ferdigstilling og arkivering av testmaterialet, testmiljøet og testinfrastrukturen for fremtidig gjenbruk.
- Overlevering av testware til de som er ansvarlige for vedlikehold.

- Analyse av erfaringer for å bestemme endringer som behøves ved fremtidige utgivelser og prosjekter.
- Bruk av informasjonen som er samlet for forbedring av testmodenhet.

1.5 Testingens psykologi (K2)

35 minutter

Begreper

Feilgjetting, uavhengighet.

Bakgrunn

Innstillingen som bør benyttes ved testing og granskning er annerledes enn den som behøves under utviklingen av programvaren. Med den rette innstillingen under testing og granskning kan utviklere teste sin egen kode, men ansvar for testen blir oftest overført til en tester. Dette blir gjort for å fokusere prosessen og for å få ytterligere fordeler, som et uavhengig synspunkt fra spesielt opplærte og profesjonelle testressurser. Uavhengig testing kan utføres på alle nivå av testen.

En viss grad av uavhengighet (for å unngå forfatterens inhabilitet) er ofte mer effektivt for å identifisere feil. Uavhengighet er derimot ikke en erstatning for inngående kjennskap, og utviklere kan ofte raskt identifisere feil i sin egen kode. Flere nivåer av uavhengighet kan spesifiseres fra lavt til høyt:

- Tester utarbeidet av den personen/ de personene som har laget programvaren som testes (lav grad av uavhengighet).
- Tester utarbeidet av en annen person/ andre personer (f. eks. fra utviklingsteamet).
- Tester utarbeidet av en person/ personer fra en annen organisasjonsgruppe (f. eks. et uavhengig testteam) eller testspesialister (f. eks. spesialisert på brukervennlighet eller ytelse).
- Tester utarbeidet av en person/ personer fra en annen organisasjon eller firma (f. eks. "outsourcing" eller sertifisering av en ekstern organisasjon).

Mennesker og prosjekter er drevet av formål. Folk pleier å tilpasse sine egne planer til de formålene som deres overordnede eller andre interesserte parter har satt opp, f. eks. å finne feil eller bekrefte at programvaren fungerer. Det er derfor viktig å formidle testingens formål på en tydelig måte.

Identifisering av feil under testingen kan oppfattes som kritikk mot produktet og mot produktets skaper. Testing oppfattes derfor ofte som en negativ aktivitet, selv om det jo er en svært konstruktiv del av behandlingen av produktrisiko. Å lete etter systemfeil i et produkt krever nysgjerrighet, profesjonell pessimisme, en kritisk innstilling, et øye for detaljer, gode kommunikasjonsevner overfor utviklerne som man samarbeider med og erfaring for å gjette hva som kan være feil. Hvis feil kommuniseres på en konstruktiv måte, kan dårlige følelser mellom testeren og analytikerne, designerne og utviklerne unngås. Dette gjelder granskning så vel som testing.

Testeren og testlederen må ha gode evner i mellommenneskelig kommunikasjon for å kunne kommunisere informasjon om feil, fremgang og risiko på en konstruktiv måte. Informasjon om feil kan hjelpe forfatteren av programvaren eller et dokument i å forbedre sine ferdigheter. Feil som blir identifisert og korrigert i testingen vil spare tid og penger senere i prosessen, samt redusere risiko.

Kommunikasjonsproblemer kan oppstå, spesielt dersom testerne kun blir sett på som budbringere av ubedt informasjon om feil og mangler. Imidlertid er det mange måter man kan forbedre kommunikasjonen og forholdene mellom testerne og andre:

- Start med samarbeid, ikke konflikt – minn alle på at de har det samme hovedmålet (et system med bedre kvalitet)
- Kommuniser resultater på en nøytral, faktafokusert måte uten å kritisere de som har skapt produktet, skriv objektive og faktabaserte hendelsesrapporter og granskningsrapporter.
- Prøv å forstå hva de andre føler og hvorfor de reagerer som de gjør.
- Sikre deg at de andre har forstått det du har sagt og at du forstår det de sier.

1.6 Etikk for testeren (K2)

10 minutter

Involvering i programvaretesting gjør det mulig for den enkelte å få tak i konfidensiell og privilegert informasjon. Etske regler er nødvendig for blant annet å sikre at informasjonen ikke brukes upassende. Ved å erkjenne ACM og IEEE sine etiske regler for ingeniører vil ISTQB® fremheve følgende etiske regler

OFFENTLIG - Sertifiserte testere skal oppføre seg i samsvar med offentlige interesser.

KLIENT OG ARBEIDSGIVER - Sertifiserte testere skal handle på en måte som er til beste for sine klienter og arbeidsgivere, og i samsvar med offentlige interesser.

PRODUKT - Sertifiserte testere skal sikre at leveransene de tilbyr (for produktet og systemet de tester) møter høyest mulige profesjonelle standarder.

VURDERING - Sertifiserte testere skal passe på integritet og uavhengighet i deres profesjonelle vurderinger.

LEDELSE - Sertifiserte testledere og ledere skal søke og fremme en etisk framgangsmåte for ledelse av programvaretesting.

PROFESJON - Sertifiserte testere skal øke integritet og anseelsen av profesjonen i samsvar med offentlige interesser.

KOLLEGER - Sertifiserte testere skal være rettfærdig og støtte deres kolleger og fremme samarbeid med programvareutviklere.

SELV - Sertifiserte testere skal delta i livslang læring vedrørende utøvelsen av sin profesjon og de skal fremme en etisk framgangsmåte til utførelsen av jobben.

Referanser

- 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 Testing gjennom livssyklusen (K2)

115 minutter

Læremål for testing gjennom programvarens livssyklus

Målene identifiserer hva du skal kunne etter å ha gjennomgått hver modul.

2.1 Utviklingsmodeller for programvare (K2)

- LO-2.1.1 Forklar sammenhengen mellom utvikling, testaktiviteter og arbeidsprodukter i utviklingslivssyklusen ved å gi eksempler basert på kontekst og egenskaper av prosjekt og produkt. (K2)
- LO-2.1.2 Erkjenne at utviklingsmodeller må tilpasses prosjektets kontekst og produktets egenskaper. (K1)
- LO-2.1.3 Huske karakteristiske egenskaper for god testing i enhver livssyklusmodell. (K1)

2.2 Testnivåer (K2)

- LO-2.2.1 Sammenligne de ulike testnivåene: hovedmål, typiske testobjekter, typiske mål for testingen (for eksempel funksjonell eller strukturell) og relaterte arbeidsprodukter, personale som tester og feiltyper som skal finnes. (K2)

2.3 Testtyper (K2)

- LO-2.3.1 Sammenligne de fire testtypene (funksjonell, ikke-funksjonell, strukturell og endringsrelatert) ved å gi eksempler. (K2)
- LO-2.3.2 Erkjenne at funksjonelle og strukturelle tester kan være del av ethvert testnivå. (K1)
- LO-2.3.3 Identifisere og beskrive ikke-funksjonelle testtyper basert på ikke-funksjonelle krav. (K2)
- LO-2.3.4 Identifisere og beskrive testtyper basert på analyse av systemstrukturen eller arkitekturen. (K2)
- LO-2.3.5 Beskrive målet med gjentatt testing og regresjonstesting. (K2)

2.4 Testing i vedlikehold (K2)

- LO-2.4.1 Sammenligne vedlikeholdstesting (testing på et eksisterende system) med testing av en ny applikasjon angående testtyper, hva som utløser testing og omfanget av testing. (K2)
- LO-2.4.2 Identifisere grunner for vedlikeholdstesting (endring, migrering og at et system blir tatt ut av bruk). (K1)
- LO-2.4.3 Beskrive rollen regresjonstesting og konsekvensanalyse har i vedlikehold. (K2)

2.1 Utviklingsmodeller for programvare(K2)	20 minutter
--	-------------

Begreper

Standardsystem, iterativ-inkrementell utviklingsmodell, validering, verifikasjon, V-modell.

Bakgrunn

Testing eksisterer ikke isolert. Testaktiviteter henger sammen med utviklingsaktivitetene. Ulike utviklingsmodeller behøver ulike teststrategier.

2.1.1 V-modellen (sekvensiell utviklingsmodell) (K2)

Selv om det finnes varianter av V-modellen, bruker en vanlig type V-modell fire testnivåer som hører sammen med de fire utviklingsnivåene.

De fire testnivåer i dette pensum er:

- komponent (enhets) testing;
- integrasjonstesting;
- systemtesting;
- akseptansetesting.

I praksis kan en V-modell ha flere, færre eller ulike nivåer av utvikling og testing. Dette avhenger av prosjekt og produkt. For eksempel kan en ha komponentintegrasjonstesting etter komponenttesting, og systemintegrasjonstesting etter systemtesting.

Programvarens arbeidsprodukter (som forretningsscenarier eller brukstilfelle (use case), kravspesifikasjoner, designdokumenter og kode) som er laget under utviklingen er ofte basis for testingen i et eller flere testnivåer. Referanser for generiske arbeidsprodukter omfatter Capability Maturity Model Integration (CMMI) eller 'Software life cycle processes' (IEEE/IEC 12207). Verifikasjon og validering (og tidlig testutvikling) kan utføres under utvikling av programvarens arbeidsprodukter.

2.1.2 Iterativ-inkrementell utviklingsmodeller (K2)

Iterativ-inkrementell utvikling er prosessen med å etablere krav, konstruere, bygge og teste et system som en serie av kortere utviklingscykluser. Eksempler er: Prototyping, rask applikasjonsutvikling (RAD), Rational Unified Process (RUP) og agile (smidige) utviklingsmodeller. Det resulterende system som er produsert i en iterasjon kan testes på flere nivåer som del av dens utvikling i iterasjonen. Et nytt delsystem som legges til de andre som er utviklet før utgjør et voksende delvis utviklet system, og det bør også testes. Regresjonstesting blir stadig mer viktig etter den første iterasjonen. Verifikasjon og validering kan utføres for hvert inkrement.

2.1.3 Testing i en livssyklusmodell (K2)

I enhver livssyklusmodell er god testing karakterisert ved følgende:

- For hver utviklingsaktivitet er det en tilsvarende testaktivitet.
- Hvert testnivå har bestemte testmål.
- Analyse og design av testene for et gitt testnivå bør starte når den tilsvarende utviklingsaktiviteten utføres.
- Testere bør delta i gjennomgang av dokumenter så snart utkast er tilgjengelig.

Testnivåer kan kombineres eller reorganiseres avhengig av prosjektets egenskaper eller systemarkitekturen. For eksempel: for å integrere et innkjøpt standardprodukt i et system kan kunden utføre integrasjonstesting på systemnivå (dvs. integrasjon med infrastrukturen og andre systemer, eller distribusjon og implementering av systemet) og akseptansetesting (funksjonelt og ikke-funksjonelt samt brukerakseptansetesting og driftsakseptansetesting).

2.2 Testnivåer (K2)**40 minutter****Begreper**

Alfatesting, betatesting, komponenttesting (også kjent som enhets-, modul- eller programtesting), driver, felttesting, funksjonelt krav, integrasjon, integrasjonstesting, ikke-funksjonelt krav, robusthetstesting, stubb, systemtesting, testnivå, testdrevet utvikling, testmiljø, brukerakseptansetesting.

Bakgrunn

For hvert testnivå kan en identifisere følgende: De generiske mål, arbeidsprodukt(ene) en bruker for å konstruere testtilfelle (testgrunnlag), testobjektet (dvs. hva en tester), typiske feil som skal finnes, krav til testramme og verktøystøtte samt spesifikke strategier og ansvarsforhold.

Å teste konfigurasjonsdata til et system skal vurderes under testplanlegging, dersom slike data er del av et system.

2.2.1 Komponenttesting (K2)

Test basis:

- Krav til komponenten
- Detaljdesign
- Kode

Typiske testobjekter:

- Komponenter
- Programmer
- Datakonverterings- / migrasjonsprogrammer
- Databasemoduler

Komponenttesting (også kjent som enhets- eller modultesting) søker feil og verifiserer programvare som kan testes separat (som moduler, programmer, objekter, klasser etc.). Dette kan skje isolert fra resten av systemet, avhengig av hvordan utviklingen er organisert og hva slags system det er. Stubber, drivere og simulatorer kan brukes.

Komponenttesting kan omfatte testing av funksjonalitet og spesifikke ikke-funksjonelle egenskaper, som ressursbruk (for eksempel minnelekkasjer) eller robusthet. Komponenttesting kan også omfatte hvit boks testing. Testtilfeller utvikles ved å analysere arbeidsprodukter som komponentspesifikasjoner, programmets design eller datamodellen.

Typisk skjer komponenttesting med tilgang til koden som testes og med støtte av utviklingsmiljøet, som et rammeverk for enhetstest eller debuggingsverktøy. I praksis blir det ofte gjort av den som skrev koden. Feil rettes vanligvis med en gang etter at de har blitt funnet uten formell feilhåndtering.

En framgangsmåte for komponenttesting er å forberede og automatisere testtilfeller før kodingen. Dette kalles test først metoden eller testdrevet utvikling. Denne metoden er i høy grad iterativ og er basert på sykluser der en først utvikler testtilfeller, og så bygger og integrerer små deler av koden, og så utfører komponenttester og så retter feil og gjentar prosessen inntil koden virker.

2.2.2 Integrasjonstesting (K2)

Testbasis:

- Software- og systemdesign
- Arkitektur
- Arbeidsflyt
- Brukstilfeller (Use cases)

Typiske testobjekter:

- Delsystemer og databaseimplementasjon
- Infrastruktur
- Grensesnitt
- Systemkonfigurasjon
- Konfigurasjonsdata

Integrasjonstesting tester grensesnitt mellom komponenter og kommunikasjon mellom ulike deler av systemet. Disse kan være operativsystem, filsystem, hardware eller grensesnitt mellom systemer.

Det kan finnes flere integrasjonstestnivåer og de kan bli utført på testobjekter av ulik størrelse. For eksempel:

1. Komponentintegrasjonstesting tester kommunikasjonen mellom komponenter og blir gjort etter komponenttesting;
2. Systemintegrasjonstesting tester interaksjonen mellom ulike systemer og mellom hardware og software og kan bli gjort etter systemtesting. I dette tilfelle kan det hende at utviklingsorganisasjonen bare styrer den ene siden av et grensesnitt. Dette kan utgjøre en egen risiko.
3. Test av forretningsprosesser kan omfatte test av en hel serie av systemer.
4. Integrasjonstesting kan også undersøke overganger mellom forskjellige plattformer.

Jo større omfanget av integrasjonen er, jo vanskeligere blir det å isolere feil til en bestemt komponent eller et bestemt system. Dette kan føre til økt risiko og ekstra tidsbruk for å håndtere trøbbel.

Systematiske integrasjonsstrategier kan være basert på systemarkitekturen (som ovenfra ned eller nedenfra opp), på gjennomgående funksjoner, transaksjonsprosessering eller andre aspekter ved komponentene eller systemet. For å lette feilisolering og for å finne feil tidlig bør en vanligvis heller integrere trinnvis enn alt på en gang ("big bang").

Integrasjonstesting kan også omfatte testing av bestemte ikke-funksjonelle egenskaper (for eksempel ytelse).

På hvert integrasjonstrinn konsentrerer testerne seg bare om integrasjonen selv. For eksempel, hvis de integrerer modul A med modul B, så er de interessert i å teste kommunikasjonen mellom modulene, ikke funksjonaliteten i den ene eller annen modul, fordi dette ble gjort i komponenttestingen. Både funksjonelle (svart boks) og strukturelle (hvit boks) strategier er aktuelle.

Det er best hvis testerne forstår arkitekturen og har innflytelse på integrasjonsleggingen. Hvis integrasjonstester er planlagt før komponentene eller systemene bygges, kan disse bygges i en bestemt rekkefølge slik at testingen blir mest mulig effektiv.

2.2.3 Systemtesting (K2)

Testbasis:

- System- og software kravspesifikasjon
- Use cases (brukstilfelle)
- Funksjonell spesifisering
- Risikoanalyserapporter

Typiske testobjekter:

- System-, bruker- og driftsmanualer
- Systemkonfigurasjon
- Konfigurasjonsdata

Systemtesting gjelder oppførselen av hele systemet eller produktet, altså hele omfanget av et prosjekt eller program. Testingens omfang skal klart uttrykkes i den overordnede testplanen og/eller planen for systemtesting.

I systemtestingen bør testomgivelsen stemme så godt som mulig overens med det endelige målmiljøet eller produksjonsmiljøet for å minimere risikoen for at miljøavhengige feil overlever testingen.

Systemtesting kan omfatte tester basert på risikoanalyser og/eller kravspesifikasjoner, forretningsprosesser, brukstilfeller (use cases) eller andre høynivåbeskrivelser eller modeller av systemets oppførsel, interaksjoner med operativsystemet og systemressurser.

Systemtesting bør omfatte både funksjonelle og ikke-funksjonelle krav til systemet og krav om datakvalitet. Krav kan eksistere som tekster og/eller modeller. Testere må også håndtere ufullstendige eller udokumenterte krav. Systemtesting av funksjonelle krav starter ved at man bruker de mest egnede spesifikasjonsbaserte (svart boks) teknikker for det aspekt ved systemet en skal teste. For eksempel kan en lage en beslutningstabell for kombinasjoner av resultater beskrevet i forretningsreglene. Strukturbaserte teknikker (hvit boks) kan etterpå brukes for å sjekke grundigheten av testingen mht. strukturelementer som en menystruktur eller navigeringen i en webside. (Se kapittel 4.)

Ofte utfører en uavhengig testgruppe systemtesten.

2.2.4 Akseptansetesting (K2)

Testbasis:

- Brukerkrav
- Systemkrav
- Brukstilfeller (Use cases)
- Forretningsprosesser
- Risikoanalyserapporter

Typiske testobjekter:

- Forretningsprosesser på det fullt integrerte systemet
- Drifts- og vedlikeholdsprosesser
- Bruksprosedyrer
- Formularer
- Rapporter
- Konfigurasjonsdata

Akseptansetesting er ofte et ansvarsområde for kundene eller brukerne av et system. Andre interessenter kan også bli involvert.

Målet med akseptansetesting er å etablere tillit til systemet, delsystemet eller spesifikke ikke-funksjonelle egenskaper av et system. Å finne feil er ikke hovedfokus i akseptansetesting. Akseptansetesting kan bedømme om et system er klart for å bli installert og brukt, selv om det ikke nødvendigvis er siste testnivå. For eksempel, en systemintegrasjonstest i stor skala kan komme etter akseptansetesten for et system

Akseptansetesting kan skje på flere ulike tider i livssyklusen, for eksempel:

- Et standardprodukt kan bli akseptansetestet når det blir installert eller integrert.
- Akseptansetesting av brukbarheten av en komponent kan bli utført under komponenttesting.
- Akseptansetesting av en ny funksjonell forbedring kan komme før systemtesting.

Typiske former for akseptansetesting omfatter de følgende:

Brukerakseptansetesting

Typisk verifiserer denne testen om systemet er klar til bruk for virkelige brukere.

Drifts(akseptanse)testing

Driftstesting i akseptansetesting av operatør og/eller administrator, inklusive:

- sikkerhetskopiering og gjenopprettelse,
- håndtering av katastrofer,
- administrasjon av brukere,
- vedlikeholdsoppgaver,
- datainnlasting og migreringsoppgaver,
- periodisk sjekk av sikkerhetshull.

Akseptansetesting vedrørende kontrakter eller forskrifter

Denne typen akseptansetesting blir gjort i forhold til en kontrakts akseptansekriterier for å produsere spesielt utviklet programvare. Akseptansekriterier bør defineres når kontrakten blir inngått.

Akseptansetesting angående forskrifter blir utført i forhold til hvilke som helst forskrifter som må oppfylles, som statlige, juridiske eller sikkerhetsmessige.

Alfa og betatesting eller felttesting

Utviklere av standardprogrammer vil ofte ønske å få tilbakemelding fra potensielle eller eksisterende kunder i deres marked før et programvareprodukt blir tilbudt for kommersielt salg. Alfatesting blir utført på utviklingsorganisasjonens sted, men ikke av utviklingsgruppen. Betatesting eller felttesting blir utført av kunder eller potensielle kunder på deres egne plattformer.

Organisasjoner kan også bruke andre betegnelser, som for eksempel “factory acceptance testing” og “site acceptance testing” for systemer som blir testet før og etter at de blir flyttet til en kundeinstallasjon.

2.3 Testtyper (K2)**40 minutter****Begreper**

Funksjonell testing, Black-box testing, svart boks testing, kodedekning, interoperabilitetstesting, belastningstesting, vedlikeholdbarhetstesting, ytelsestesting, portabilitetstesting, pålitelighetstesting, sikkerhetstesting, spesifikasjonsbasert testing, stresstesting, brukbarhetstesting, strukturell testing, white-box testing, hvit boks testing.

Bakgrunn

En gruppe testaktiviteter kan ha som mål å verifisere programsystemet (eller delen av et system) med grunnlag i en bestemt begrunnelse eller et bestemt testmål.

En testtype er fokusert på et bestemt testmål, som kan være test av en funksjon, som programmet skal utføre eller en ikke-funksjonell kvalitetsegenskap, som for eksempel pålitelighet eller brukbarhet, strukturen eller arkitekturen til programmet eller systemet, eller relatert til endringer, dvs. sikring at feil har blitt rettet (re-testing) og kontroll mot utilsiktede bivirkninger (regresjonstesting).

Man kan utvikle eller bruke en modell av programvaren for strukturell testing (for eksempel en kontrollflytmodell eller modell av menystrukturen), ikke-funksjonell testing (for eksempel en modell for ytelse eller brukervennlighet, en trusselmodell for sikkerhet), og for funksjonell testing (for eksempel en prosessflytmodell, tilstandsmoell eller en spesifikasjon i naturlig språk).

2.3.1 Testing av funksjonalitet (funksjonell testing) (K2)

Funksjonen et system, delsystem eller en komponent skal utføre kan være beskrevet i arbeidsprodukter som for eksempel en kravspesifikasjon, brukstilfelle (use case), eller en funksjonell spesifikasjon, eller de kan være udokumentert (underforstått). Funksjonene er "hva" systemet gjør.

Funksjonelle tester er basert på funksjoner og "features" (beskrevet i dokumenter eller kjent av testerne) og deres interaksjon med spesifikke systemer, og kan utføres på alle testnivåer (tester for komponenter kan for eksempel baseres på komponentspesifikasjonen).

Spesifikasjonsbaserte teknikker kan brukes for å utlede testbetingelser og testtilfelle fra programvarens eller systemets funksjonalitet. (Se kapittel 4). Funksjonell testing er opptatt av den eksternt synlige oppførselen av programvaren (svart boks testing).

En type funksjonell testing, sikkerhetstesting, undersøker funksjonene (/for eksempel en brannmur) relatert til oppdagelsen av trusler, som for eksempel virus og angrep fra ondsinnede utenforstående. En annen type funksjonell testing, testing av interoperabilitet, evaluerer programvareproduktets evne til å samvirke med en eller flere spesifiserte komponenter eller systemer.

2.3.2 Testing av ikke-funksjonelle egenskaper av programvaren (ikke-funksjonell testing) (K2)

Ikke-funksjonell testing omfatter, men er ikke begrenset til, ytelsestesting, belastningstesting, stresstesting, brukbarhetstesting, vedlikeholdbarhetstesting, pålitelighetstesting og portabilitetstesting. Dette er testing av "hvordan" systemet virker.

Ikke-funksjonell testing kan utføres på alle testnivåer. Begrepet ikke-funksjonell testing beskriver testene som kreves for å måle egenskaper av systemer og programvare som kan kvantifiseres på en variabel skala, som for eksempel svartider for ytelsestesting. Disse testene kan referere til en kvalitetsmodell som den som er definert i standarden 'Software Engineering – Software Product Quality' (ISO 9126-1). Ikke-funksjonell testing ser på den eksternt synlige oppførselen av programvaren og bruker for det meste svart boks teknikker for testdesign.

2.3.3 Testing av programvarens struktur/arkitektur (strukturell testing, hvit boks testing) (K2)

Strukturell (hvit boks) testing kan utføres på alle testnivå. Strukturelle teknikker blir best brukt etter spesifikasjonsbaserte teknikker, for hjelp til å måle grundigheten av testingen. Dette gjøres ved å bedømme deknningen av diverse typer struktur.

Dekningsgraden for en struktur som er blitt testet, uttrykkes ved andel fullførte tester i prosent av alle de mulige testene for det valgte dekningskriteriet. Dekningsgraden er altså graden som en struktur har blitt utført gjennom en testsuite, uttrykt som en prosent. Hvis deknningen ikke er 100%, kan en lage flere tester for å teste de ting som manglet, og dermed øke dekningsgraden. Slike teknikker er beskrevet i kapittel 4.

En kan bruke verktøy for å måle dekningsgraden mot kode av de elementene som testes, på alle testnivåer, men spesielt i komponenttestingen og i komponentintegrasjonstesting. Eksempel på slike elementer er programinstruksjoner og forgreninger. Strukturell testing kan også baseres på arkitekturen av et system, som for eksempel kallhierarkiet.

Strukturelle testteknikker kan også brukes på nivåene systemtest, systemintegrasjonstest og akseptansetest (for eksempel mot modeller av forretningsgangen eller menystrukturer).

2.3.4 Testing relatert til endringer (re-testing og regresjonstesting) (K2)

Etter at en feil er funnet og rettet, bør programvaren re-testes for å bekrefte at den opprinnelige feilen er rettet riktig. Dette kalles for bekreftelse (re-testing). Debugging (feilretting) er en utviklingsaktivitet, ikke en testaktivitet.

Regresjonstesting er ny testing av et tidligere testet program etter modifikasjoner. Dette gjøres for å sikre at feil ikke er blitt introdusert i uendrede områder i produktet. Regresjonstest tjener også til å oppdage feil som lå skult i uendrede områder i produktet, men som blir tilgjengelige etter endringen. Regresjonstest blir utført etter endringer i programvaren eller omgivelsen. Disse feilene kan enten være i programvaren som testes eller i en annen relatert eller urelatert programvarekomponent. Regresjonstesting utføres når programvaren eller dens omgivelse blir endret. Omfanget av regresjonstesting baseres på risikoen for å ikke finne feil i programvaren som virket før.

Tester bør være mulig å gjenta hvis de skal brukes til re-testing og for å hjelpe regresjonstesting.

Regresjonstesting kan utføres på alle testnivåer og omfatter funksjonell, ikke-funksjonell og strukturell testing. Regresjonstestsuiter kjøres mange ganger og utvikler seg generelt langsomt. Dette betyr at regresjonstesting er en sterk kandidat for automatisering.

2.4 Testing i vedlikehold (K2)	15 minutter
---------------------------------------	--------------------

Begreper

Konsekvensanalyse, vedlikeholdstesting.

Bakgrunn

Når et programvaresystem er satt i drift, er det ofte i drift for år eller tiår. I løpet av denne tiden blir systemet, dets konfigurasjonsdata eller dets omgivelse ofte korrigeret, endret eller utvidet. For suksess i vedlikeholdstesting er det avgjørende at frigivelser blir planlagt på forhånd. En må skille mellom planlagte frigivelser og nødsendringer. Vedlikeholdstesting blir gjort på et eksisterende system i drift, og blir utløst av endringer i omgivelsen, endringer, migrasjon eller avvikling av programvaren eller systemet.

Endringer omfatter planlagte forbedringer (for eksempel ved nye frigivelser), korreksjoner og nødsendringer av programvaren eller systemet. Det omfatter også endringer i omgivelsen slik som planlagte oppgraderinger av operativ- og databasesystem, planlagte oppgraderinger av standardsystemer, eller patcher for å korrigere nye sårbarheter av operativsystemet som er funnet eller blitt kjent.

Vedlikeholdstesting for migrasjon (for eksempel fra en plattform til en annen) bør omfatte driftstester av den nye omgivelsen og av den endrete programvaren. Migrasjonstesting (konverteringstesting) trengs også når data fra en annen applikasjon skal migreres inn i systemet som vedlikeholdes.

Vedlikeholdstesting for avvikling av et system kan omfatte testing av datamigrasjon eller arkivering hvis data må oppbevares i lang tid.

I tillegg til testing av det som er blitt endret, omfatter vedlikeholdstesting omfattende regresjonstesting av de deler av systemet som ikke er blitt endret. Omfanget av vedlikeholdstesting henger sammen med risikoen med endringen, størrelsen på det eksisterende systemet og størrelsen på endringen. Avhengig av endringene kan vedlikeholdstesting gjøres på et eller alle testnivåer og for en eller alle testtyper. Å avgjøre hvordan endringer kan ha følger for det eksisterende system, kalles konsekvensanalyse. Denne brukes til å beslutte hvor mye regresjonstesting en skal gjøre. Konsekvensanalyse kan brukes for å avgjøre hva som skal være med i en regresjonstestsuite.

Vedlikeholdstesting kan være vanskelig hvis spesifikasjonene er avleggs, de ikke finnes eller en ikke har testere som har kunnskap til bruksområdet.

Referanser

- 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 829-1998
- 2.4 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE 829-1998

3 Statistiske teknikker (K2)	60 minutter
-------------------------------------	--------------------

Læremål for statistiske teknikker

Læremålene identifiserer hva du skal være i stand til å gjøre når du har gjennomgått hver modul.

3.1 Statistiske teknikker og testprosessen (K2)

- LO-3.1.1 Gjenkjenne arbeidsresultater ved programvare som kan sjekkes ved ulike statistiske teknikker. (K1)
- LO-3.1.2 Beskrive viktigheten og verdien av statistiske teknikker for bedømmelse av arbeidsprodukter ved programvare. (K2)
- LO-3.1.3 Forklare forskjellen mellom statistiske og dynamiske teknikker ved å vurdere målsetningene, typer av feil som kan identifiseres, og rollen av disse teknikkene i livssyklusen av programvare. (K2)

3.2 Granskningsprosessen (K2)

- LO-3.2.1 Kjenne fasene, roller og ansvarsforhold ved en typisk formell granskning. (K1)
- LO-3.2.2 Forklare forskjellen mellom ulike typer granskninger: uformell granskning, teknisk gjennomgang, walkthrough og inspeksjon. (K2)
- LO-3.2.3 Forklare faktorene for suksessfull utføring av granskninger. (K2)

3.3 Statisk analyse med verktøy (K2)

- LO-3.3.1 Gjenkjenne typiske feil som identifiseres av statisk analyse og sammenlikne dem med granskninger og dynamisk testing. (K1)
- LO-3.3.2 Beskrive ved eksempler typiske nytteeffekter av statisk analyse. (K2)
- LO-3.3.3 Liste opp typiske kode- og designfeil som kan identifiseres ved bruk av verktøy for statisk analyse. (K1)

3.1 *Statistiske teknikker og testprosessen (K2)*

15 minutter

Begreper

Dynamisk testing, statisk testing.

De to ordene granskning og review brukes i samme betydning.

Bakgrunn

Til forskjell fra dynamisk testing som krever utføring av programmet, baserer statistiske testteknikker seg på manuell kontroll (reviews, granskninger) og automatisk analyse (statisk analyse) av koden eller annen prosjektdokumentasjon uten at koden blir utført.

Granskning er en måte å teste arbeidsprodukter ved programvare (inklusive kode) og kan utføres lenge før en utfører dynamisk test. Feil funnet i granskninger tidlig i livssyklusen er ofte mye billigere å rette enn de en oppdager når en utfører tester (for eksempel feil funnet i kravene).

En granskning kan utføres helt manuelt, men også med verktøystøtte. Den i hovedsak manuelle aktiviteten er å kontrollere et arbeidsprodukt og kommentere det. Hvilket som helst arbeidsprodukt kan gjennomgås, inklusive kravspesifikasjoner, designspesifikasjoner, kode, testplaner, testspesifikasjoner, testtilfelle, testskripter, brukermanualer eller websider.

Fordeler med granskninger omfatter at en finner og retter feil tidlig, at produktiviteten i utviklingen øker, utviklingen går fortere, og at testingens kostnad og tidsbruk minsker. Kostnaden i løpet av hele livstiden til et program blir redusert, det blir færre feil og forbedret kommunikasjon. Granskninger kan finne uteglemte ting, for eksempel i krav, som er vanskelig å finne i dynamisk testing.

Granskninger, statisk analyse og dynamisk testing har same mål – å identifisere feil. Teknikkene utfyller hverandre: de ulike teknikkene kan finne ulike typer feil på en effektiv og produktiv måte. Sammenlignet med dynamisk testing, finner statistiske teknikker årsaken til problemer (altså feilen selv), ikke bare symptomene.

Typiske feil som er lettere å finne i granskninger enn i dynamisk testing er: avvik fra standarder, feilaktige krav, feil i design, utilstrekkelig vedlikeholdbarhet og feilaktige grensesnittspesifikasjoner.

3.2 Granskningsprosessen (K2)

25 minutter

Begreper

Inngangskriterier, formell granskning, uformell granskning, inspeksjon, måling, moderator/inspeksjonsleder, peer review, revisor, reviewer, protokollant, teknisk gjennomgang, walkthrough.

Begrepene strukturert gjennomgang og walkthrough er brukt i samme betydning.

Bakgrunn

De ulike typer granskninger varierer fra meget uformell (for eksempel ingen nedskrevne instruksjoner for de som gjennomgår) til meget formell (dvs. inkluderer medvirkningen av hele gruppen, dokumenterte resultater og dokumenterte prosedyrer for å holde en granskning). Formaliteten til en granskningsprosess henger sammen med faktorer som utviklingsprosessens modenhet, krav i lover eller forskrifter eller behovet for sporbarhet.

Måten en granskning utføres på avhenger av målet som en er blitt enig om for granskningen (dvs. finne feil, forstå problemområdet, opplæring av testere eller nye medarbeidere eller diskusjon og beslutning ved hjelp av konsensus).

3.2.1 Faser i en formell granskning (K1)

(Dette avsnitt avviker fra den engelske versjonen).

En typisk formell granskning har følgende hovedfaser:

1. Planlegging:
 - Definere granskningskriterier
 - velge personene
 - tildele roller
 - definere start- og sluttkriterier for mer formelle granskninger (som inspeksjoner)
 - velge hvilke deler av dokumentet en skal se på
2. Oppstart:
 - dele ut dokumentene
 - forklare målsetningene, prosess og dokumentene til deltakerne
 - sjekke inngangskriteriene (for mer formelle granskningstyper).
3. Individuell forberedelse:
 - arbeid som gjøres av hver deltaker alene før granskningsmøte
 - forberede seg for granskningsmøtet ved å gjennomgå dokumentet / dokumentene
 - skrive ned mulige feil, spørsmål og kommentarer.
4. Granskningsmøte:
 - diskutere eller loggføre, med dokumenterte resultater eller protokoll (for mer formelle granskningstyper).
 - notere feil, gjøre anbefalinger for å håndtere dem eller gjøre beslutninger om feilene.
 - gjennomgang, evaluering og nedskrivning av anmerkninger. Dette kan skje i gruppemøtet eller gjennom elektroniske midler.
5. Retting:
 - rette de feil som er funnet, vanligvis gjort av forfatteren.
 - loggføre endret status av feilene (i formelle granskninger)
6. Oppfølging:
 - kontrollere at feilene er gjort noe med
 - samle måldata
 - sjekke at sluttkriteriene er oppfylt (for mer formelle granskningstyper).

3.2.2 Roller og ansvar (K1)

En typisk formell granskning har følgende roller:

- **Manager:** Beslutter om granskninger skal gjøres, setter av tid i prosjektplaner og bestemmer om målene til en granskning er blitt oppnådd.
- **Granskningsleder / Moderator:** Personen som leder granskningen av dokumentet eller samlingen dokumenter. Dette omfatter planlegging av granskningen, møteledelse og oppfølging etter møtet. Hvis nødvendig kan moderatoren megle mellom de ulike synspunktene. Moderatoren er ofte avgjørende for suksessen av en granskning.
- **Forfatter:** Den som har skrevet dokumentet (eller dokumentene) eller har hovedansvaret for det som skal gjennomgås.
- **Reviewere / Revisorer:** Personer med spesifikk teknisk eller anvendelsesbakgrunn (de blir også kalt kontrollører eller inspektører) som, etter nødvendig forberedelse, identifiserer og beskriver saker de finner (for eksempel feil) i produktet som gjennomgås. Revisorer bør velges slik at de kan representere ulike perspektiver og roller i granskningsprosessen, og bør delta i granskningsmøtene.
- **Protokollant (eller sekretær):** Dokumenterer alle anmerkninger, problemer og åpne punkter som blir identifisert i møtet.

Å se på programvareprodukter eller relaterte arbeidsresultater (dokumenter) fra ulike perspektiver og å bruke sjekklister kan gjøre granskninger mer effektive. For eksempel: en sjekkliste basert på ulike perspektiver som bruker, vedlikeholdsansvarlig, tester eller driftsansvarlig, eller en sjekkliste bestående av typiske problemer med krav, kan hjelpe til med å finne hittil oversette problemer.

3.2.3 Granskningstyper (K2)

Et enkelt dokument kan utsettes for mer enn en granskning. Hvis mer enn en type granskning blir brukt, kan rekkefølgen på granskningene variere. For eksempel kan en først utføre en uformell granskning, så en teknisk gjennomgang. En inspeksjon kan utføres på en kravspesifikasjon før en walkthrough med kunder. Hovedegenskapene, mulighetene og målsetningene av de vanlige granskningstyper er:

Uformell granskning

Hovedegenskaper:

- Ingen formell prosess;
- Dette kan være parprogrammering eller at en teknisk ekspert går gjennom design og kode;
- Kan (men må ikke) være dokumentert;
- Nytteverdien kan variere avhengig av hvem som gjennomgår;
- Hovedformål: En billig måte å få nyttig feedback.

Strukturert gjennomgang / Walkthrough

Hovedegenskaper:

- Møte ledes av forfatteren;
- Scenarier, "tørrkjøring", gruppe av arbeidskolleger;
- Møtet har åpen dagsorden;
- Mulighet for at deltakerne forbereder seg før møtet, og mulighet for granskningsrapport, liste av hva som er funnet og mulighet for en protokollant (som ikke er forfatter);
- Kan i praksis variere fra meget uformell til meget formell;
- Hovedformål: Å lære, å få forståelse for dokumentet, å finne feil.

Teknisk gjennomgang

Hovedegenskaper:

- Dokumentert, definert prosess for å finne feil som inkluderer arbeidskolleger og tekniske eksperter med mulighet for deltakelse til ledelsen;
- Kan utføres som en "peer review" uten at ledelsen deltar;
- I idealtilfelle ledet av en opplært møteleder (moderator), ikke av forfatteren;
- Reviewerne forbereder seg før møtet;
- Valgfri bruk av sjekklister
- Utarbeidelse av en granskningsrapport som omfatter en liste av hva som er funnet og utsagn om produktet oppfyller sine krav og, der det behøves, anbefalinger vedrørende de ting som er funnet.
- Kan i praksis variere fra meget uformell til meget formell;
- Hovedformål: Å diskutere, gjøre beslutninger, evaluere alternativer, finne feil, løse tekniske problemer og sjekke samsvar med spesifikasjoner, planer, forskrifter og standarder.

Inspeksjon

Hovedegenskaper:

- Ledet av en opplært møteleder (moderator), ikke av forfatteren;
- Vanligvis sjekking ved hjelp av arbeidskolleger;
- Definerte roller;
- Samling og bruk av måldata;
- Formell prosess basert på regler og sjekklister;
- Spesifiserte inngangs- og utgangskriterier for å godkjenne produktet;
- Forberedelse før møtet;
- Inspeksjonsrapport med liste av anmerkninger;
- Formell oppfølgingsprosess;
- Valgfri bruk av data til prosessforbedring;
- Valgfri bruk av en leser;
- Hovedformålet er å finne feil.

Walkthroughs, tekniske gjennomganger og inspeksjoner kan utføres i en gruppe av kolleger på samme organisatoriske nivå. Denne typen granskning kalles kollegagjennomgang.

3.2.4 Suksessfaktorer for granskninger (K2)

Suksessfaktorer for granskninger omfatter:

- Hver granskning har et klart og på forhånd definert mål.
- De rette mennesker for disse målsetninger er med.
- Testerne blir verdsatt som deltakere som bidrar til en review og lærer om produktet, noe som gjør det mulig for dem å forberede testene tidligere.
- Feilene som finnes er velkomne og de rapporteres objektivt.
- Problemer mht. involverte personer og psykologiske aspekter blir håndtert på rett måte (dvs. at en granskning blir til en positiv opplevelse for forfatterne og de andre deltakere).
- Granskningsteknikker som er egnet for å oppnå målsetningen og type og nivå av programvareprodukter og deltakere blir brukt.
- Sjekklister eller roller brukes hvis passende for å øke effektiviteten med å finne feil.
- Deltakerne er opplært i granskningsteknikker, spesielt når det gjelder de mer formelle teknikker, som inspeksjon.
- Ledelsen støtter en god granskningsprosess (for eksempel ved å avsette nok tid i prosjektplaner for granskningsaktiviteter).
- Det legges vekt på læring og prosessforbedring.

3.3 Statisk analyse med verktøy (K2)**20 minutter****Begreper**

Kompilator, kompleksitet, kontrollflyt, dataflyt, statistisk analyse.

Bakgrunn

Målet med statistisk analyse er å finne feil i kildekode og modeller for programvaren. Statistisk analyse gjøres uten å utføre det som analyseres; det analyseres med verktøy. Dynamisk testing utfører koden. Statistisk analyse kan lokalisere feil som er vanskelige å finne i (dynamisk) testing. Som ved granskninger finner statistisk analyse feilene direkte istedenfor via symptomer. Statistiske analyseverktøy analyserer programkode (dvs. kontroll- og dataflyt), men også generert output som HTML og XML.

Verdien av statistisk analyse er:

- Tidlig oppdagelse av feil før testutføring.
- Tidlig varsel om mistenkelige egenskaper ved koden eller designet ved å beregne måleverdier, som høy kompleksitet.
- Identifisering av feil som dynamisk testing ikke finner så lett.
- Oppdagelse av avhengigheter og selvmotsigelser i software-modeller; for eksempel linker.
- Forbedret vedlikeholdbarhet av kode og design.
- Forebygging av feil, hvis en lærer av feilene under utvikling.

Typiske feil som statistiske analyseverktøy finner omfatter:

- Å referere en variabel med en udefinert verdi;
- Avvik i grensesnitt mellom moduler og komponenter;
- Variabel som aldri brukes eller er feildeklart;
- Kode som ikke kan utføres (død kode);
- Manglende og feil logikk, for eksempel uendelige løkker;
- For kompliserte konstruksjoner;
- Avvik fra programmeringsstandarder;
- Sårbarheter (sikkerhet);
- Feil syntaks på kode og programvaremodeller.

Statistiske analyseverktøy blir typisk brukt av utviklere (for å sjekke mot forhåndsdefinerte regler og programmeringsstandarder) før og under komponent- og integrasjonstesting eller når de sjekker inn kode i konfigurasjonsstyringsverktøy, og av designere under modellering av programvaren. Statistiske analyseverktøy kan produsere et stort antall advarsler, og disse må styres godt for at verktøyene kan bli brukt effektivt.

Kompilatorer kan også støtte statistisk analyse, inklusive beregning av måleverdier.

Referanser

- 3.2 IEEE 1028
- 3.2.2 Gilb, 1993, van Veenendal, 2004
- 3.2.4 Gilb, 1993, IEEE 1028
- 3.3 Van Veenendal, 2004

4 Teknikker for testdesign (K3)

285 minutter

Læremål for teknikker for testdesign.

Læremålene viser hva du skal kunne etter fullførelsen av hver modul.

4.1 Testutviklingsprosessen. (K3)

LO-4.1.1 Skill mellom en testdesign spesifikasjon, testtilfelle spesifikasjon og testprosedyre. (K2)

LO-4.1.2 Sammenligne begrepene testbetingelse, testtilfelle og testprosedyre. (K2)

LO-4.1.3 Evaluere kvaliteten på testtilfeller. Inneholder de:

- tydelig sporbarhet til kravene
- forventet resultat (K2)?

LO-4.1.4 Oversette testtilfeller til en velstrukturert testprosedyre på et detaljnivå tilpasset testernes kunnskaper. (K3)

4.2 Kategorier av teknikker for testdesign (K2)

LO-4.2.1 Begrunn at både spesifikasjonsbaserte (svart boks) og strukturbaserte (hvit boks) angrepsvinkler er nyttige til design av testtilfeller, samt nevnt de vanligste teknikkene for hver av dem. (K1)

LO-4.2.2 Forklar egenskapene til og likhetene og forskjellene mellom spesifikasjonsbasert testing, strukturbasert testing og erfaringsbasert testing. (K2)

4.3 Spesifikasjonsbaserte eller svart boks teknikker (K3)

LO-4.3.1 Skriv testtilfeller fra en gitt software modell ved å bruke følgende testteknikker: (K3)

- ekvivalensklasseinndeling
- grenseverdianalyse
- beslutningstabelltesting
- tilstandsbasert testing

LO-4.3.2 Forklar hovedmålet med hver av de fire teknikkene, hvilket nivå og type testing som kan bruke teknikken, og hvordan dekning kan måles. (K2)

LO-4.3.3 Forklar hovedpunktene i og fordelene med brukstilfelletesting (use case testing). (K2)

4.4 Strukturbaserte eller hvit boks teknikker (K4)

LO-4.4.1 Beskriv hovedpunktene i og verdien av kodedekning. (K2)

LO-4.4.2 Forklar hovedpunktene i programinstruksjonsdekning og beslutningsdekning, samt forstå at disse teknikkene også kan bli brukt på andre testnivåer enn komponenttesting (f. eks på forretningsprosedyrer på systemnivå). (K2)

LO-4.4.3 Skriv testtilfeller fra gitte kontrollflyt ved hjelp av å bruke følgende testteknikker:

- programinstruksjonstesting
- beslutningstesting. (K3)

LO-4.4.4 Vurder dekningsgraden til programinstruksjons- og beslutningsdekning i.hht. hvor fullstendig testingen er i forhold til definerte sluttkriterier. (K4)

4.5 Erfaringsbasert teknikk (K2)

LO-4.5.1 Gjengi grunner for å skrive testtilfeller basert på intuisjon, erfaring og kjennskap til vanlige feil. (K1)

LO-4.5.2 Sammenlign erfaringsbasert teknikk med spesifikasjonsbaserte test teknikker. (K2)

4.6 Valg av test teknikker (K2)

LO-4.6.1 Klassifiser testteknikker mht. hvor egnet de er for en gitt kontekst, som f. eks. for en testbasis, forskjellige modeller og programvareegenskaper. (K2)

4.1 Testutviklingsprosessen (K3)

15 minutter

Begreper

Spesifikasjon av testtilfelle, testdesign, kjøreplan, testprosedyre, testskript, sporbarhet.

Bakgrunn

Testutviklingsprosessen som er beskrevet i denne seksjonen kan utføres på forskjellige måter, fra meget uformell med lite eller ingen dokumentasjon, til meget formell (som beskrevet under). Formalitetsnivået er avhengig av testingens kontekst, inkludert modenhetsnivået på testingen og utviklingsprosessen, tidsbegrensninger, sikkerhetskrav eller myndighetskrav og de involverte personene.

Under testanalysen analyseres dokumentasjonen av testgrunnlaget for å avklare hva som skal testes, dvs. identifisere testbetingelsene. En testbetingelse er definert som et objekt eller del av objekt som kan bli verifisert med et eller flere testtilfeller (dvs. en funksjon, transaksjon, kvalitetsegenskap eller strukturelement).

Etablering av sporbarhet fra testtilfellet tilbake til spesifisering og krav muliggjør både effektiv konsekvensanalyse når krav endres, samt mulighet til å fastslå dekning av krav for et sett med tester. Under testanalysen blir den detaljerte angrepsvinkelen for testen implementert, for å velge den eller de testdesign teknikkene som skal brukes basert på bl.a. identifisert risiko (Se kapittel 5 for mer om risikoanalyse).

Under testutviklingen blir testtilfeller og testdata skapt og spesifisert. Et testtilfelle består av et sett med inputverdier, krav til forhåndstilstand, forventede resultater og krav til avslutningstilstand utviklet for å dekke utvalgte testmål eller testbetingelse(r). IEEE 829-1998 standarden ("Standard for Software Test Documentation") beskriver innholdet av testdesignspesifikasjoner (inkludert testbetingelser) og testtilfellespesifisering.

Forventede resultater bør produseres som en del av spesifiseringen til et testtilfelle og inkludere utdata (output), endringer på data og status, og alle andre konsekvenser av testen. Hvis forventet resultat ikke er definert kan et plausibelt men feilaktig resultat bli tolket som det korrekte. Forventet resultat skal ideelt være definert før testutførelsen.

Under testimplementasjonen blir testtilfeller utviklet, implementert, prioritert og organisert i en testprosedyrespesifisering (IEEE 829-1998). Testprosedyren (eller manuelt testskript) spesifiserer rekkefølgen av punkter for utførelsen av en test. Dersom tester kjøres med bruk av automatiserte verktøy, er sekvensen av punkter spesifisert i testskript (som er en automatisert testprosedyre).

De forskjellige testprosedyrene og automatiserte testskripter blir deretter samlet i en kjøreplan, som definerer den rekkefølgen de forskjellige testprosedyrene og eventuelle automatiserte testskripter blir kjørt, når de skal kjøres og av hvem. Kjøreplanen må ta høyde for faktorer som regresjonstester, prioritering og tekniske og logiske avhengigheter.

4.2 Kategorier for teknikker for testdesign (K2)	<i>15 minutter</i>
---	--------------------

Begreper

Svart boks teknikk, erfaringsbasert teknikk, spesifikasjonsbasert teknikk, strukturell teknikk (hvit boks teknikk).

Bakgrunn

Hensikten med en testdesignteknikk er å identifisere testbetingelser, testtilfeller og testdata.

Man omtaler vanligvis testdesignteknikker som svart boks eller hvit boks (strukturell). Svart boks teknikker (også kalt spesifikasjonsbaserte teknikker) er en måte å identifisere og velge testbetingelser eller testtilfeller basert på en analyse av dokumentasjonen av det som skal testes. Det kan omfatte både funksjonell og ikke-funksjonell testing. Svart boks teknikker bruker ifølge sin definisjon ikke noe informasjon om den interne strukturen for en komponent eller et system som skal testes. Hvit boks teknikker (også kjent som strukturelle eller strukturbaserte teknikker) er basert på en analyse av strukturen til komponenten eller systemet. Både svart boks og hvit boks teknikker kan også ta i bruk erfaringene til utviklerne, testerne og brukerne for å bestemme hva som skal testes.

Noen teknikker er tydelig innen kun en kategori; andre har elementer fra begge kategoriene.

Dette pensum refererer til spesifikasjonsbaserte angrepvinkler som svart boks teknikker og strukturbaserte som hvit boks teknikker. I tillegg dekker det erfaringsbaserte teknikker.

Vanlige egenskaper for spesifikasjonsbaserte teknikker omfatter:

- Modeller, enten formelle eller uformelle, brukes for spesifisering av problemet som skal løses, programvaren eller dens komponenter.
- Testtilfeller kan systematisk bli tatt ut fra disse modellene.

Vanlige egenskaper for strukturbaserte teknikker omfatter:

- Informasjon om hvordan programvaren er konstruert er brukt for å ta ut testtilfeller, for eksempel fra kode og detaljdesign.
- Dekningsgrad av programvaren kan måles for eksisterende testtilfeller, og ekstra testtilfeller kan planlegges for systematisk å øke dekningsgraden.

Vanlige egenskaper for erfaringsbasert teknikker omfatter:

- Kunnskap og erfaring til mennesker brukes for å definere testtilfeller.
- Kunnskapen testere, utviklere, brukere og andre interessenter har om programvaren, dens bruk og miljø er en informasjonskilde.
- Kunnskap om sannsynlige feil og hvor de befinner seg er en annen informasjonskilde.

4.3 Spesifikasjonsbaserte (svart boks) teknikker (K3)	150 minutter
---	--------------

Begreper

Grenseverdianalyse, beslutningstabelltesting, ekvivalensklasseinndeling, tilstandsbasert testing, brukstilfelle (use case) testing.

Ordene brukstilfelle og use case er brukt i samme betydning.

4.3.1 Ekvivalensklasseinndeling (K3)

Input til programvaren eller systemet blir delt inn i grupper som forventes å ha lik oppførsel, og dermed sannsynligvis blir prosessert på samme måte. Ekvivalensklasser kan defineres for både gyldige data og ugyldige data, dvs. verdier som forventes godtatt hhv. å bli forkastet. Ekvivalensklasser kan også identifiseres for utdata (output), interne verdier, tidsrelaterte verdier (f. eks før eller etter en hendelse) og for grensesnittparametere (altså under integrasjonstesting). Tester kan utvikles for å dekke alle gyldige og ugyldige ekvivalensklasser. Ekvivalensklasseinndeling kan benyttes på alle nivåer av testing.

Ekvivalensklasseinndeling som en teknikk kan brukes for å oppnå dekningsmål på inndata (input) og utdata (output). Den kan benyttes for menneskelig input, input via grensesnitt mot systemet, eller grensesnittparametere under integrasjonstesting.

4.3.2 Grenseverdianalyse (K3)

Behandling av data som befinner seg i grenseområdene til ekvivalensklassene har større sannsynlighet for å være feil, så grenseverdier er et område der testing sannsynligvis vil finne feil. Maksimum og minimum verdier av en klasse er den klassens grenseverdier. En grenseverdi for en gyldig klasse er en gyldig grenseverdi; grenseverdien til en ugyldig klasse er en ugyldig grenseverdi. Tester kan utvikles for å dekke både gyldige og ugyldige grenseverdier. Når man utvikler testtilfeller, velger man en test for hver grenseverdi.

Grenseverdianalyse kan benyttes på alle testnivåer. Det er relativt enkelt å bruke og har høy sannsynlighet for å finne feil. Detaljerte spesifikasjoner er verdifulle for å bestemme interessante grenser.

Denne teknikken er ofte ansett som en utvidelse av ekvivalensklasseteknikken eller andre svart boks teknikker. Den kan brukes på ekvivalensklasser for brukerinput på skjerm i tillegg til f.eks. på tidsintervaller (f. eks tidsavbrudd, krav til transaksjonshastighet) eller tabellintervaller (f. eks. tabellens størrelse er 256*256).

4.3.3 Beslutningstabelltesting (K3)

Beslutningstabeller er en god måte å fange systemkrav som inneholder logiske valg, og for å dokumentere intern systemdesign. De kan bli brukt til å beskrive komplekse foretningsregler som et system skal implementere. Spesifikasjonen blir analysert og betingelsene og handlingene blir oftest beskrevet på en slik måte at de kan enten være sann eller usann. Beslutningstabellen inneholder det utløsende valget, ofte kombinasjoner av sant og usant for alle inputvalg og de resulterende funksjonene for hver kombinasjon av valg. Hver kolonne av tabellen tilsvarer en foretningsregel som definerer en unik kombinasjon av valg, som resulterer i utførelsen av de funksjonene som er forbundet med den spesifikke regelen. Standarden for dekningsgrad vanligvis brukt for beslutningstabelltesting er å ha minst en test per kolonne, som typisk medfører at en dekker alle kombinasjoner av utløsende valg.

Styrken til beslutningstabelltesting er at den lager kombinasjoner av valg som kanskje ellers ikke ville blitt benyttet under testing. Den kan benyttes i alle situasjoner der programvarens funksjon er avhengig av flere logiske beslutninger.

4.3.4 Tilstandsbasert testing (K3)

Et system kan oppføre seg forskjellig avhengig av nåværende tilstand eller tidligere historie (tilstanden). I dette tilfellet kan dette aspektet av systemet illustreres som et tilstandsdiagram. Det gjør at testerene kan vurdere programvaren sett ut fra dens tilstand, overgang mellom tilstander, inputverdier eller hendelser som medfører tilstandsendringer (overganger) og de stegene som kan resultere fra disse overganger. Tilstandene til systemet eller objektet under test er individuelle, identifiserbare og endelige i antall. Et tilstandsdiagram viser forholdet mellom tilstand og input, og kan belyse mulige ugyldige overganger. Tester kan konstrueres for å dekke en typisk sekvens av tilstander, for å dekke hver eneste tilstand, for å utføre hver overgang, for å utføre spesielle sekvenser av overganger eller for å teste ugyldige overganger.

Tilstandsbasert testing er mye benyttet innen innebygd programvare og generelt innen teknisk automatisering. Men teknikken er også egnet for å modellere et foretningsområde som har spesifikke tilstander eller for å teste flyten i en skjermdialog (f.eks. for internettapplikasjoner eller forretningssituasjoner)

4.3.5 Brukstilfelle (use case) testing (K2)

Tester kan bli spesifisert på bakgrunn av brukstilfelle (use case). Et brukstilfelle (use case) beskriver samhandlingen mellom deltagere, inkludert brukere og systemet, som produserer et resultat av verdi til en systembruker eller kunde. Brukstilfelle kan beskrives enten på et abstrakt nivå: Forretningstilfelle, uten å nevne teknologi, forretningsflytnivå eller på systemnivå: Brukstilfelle for bruken av systemet på nivået til systemets funksjonalitet. Hvert brukstilfelle har forhåndsbedingungen (forutsetninger) som må oppfylles for at brukstilfellet skal virke rett. Hvert brukstilfelle avsluttes med en avslutningstilstand som er det observerbare resultatet og slutttilstanden til systemet etter at brukstilfellet er utført. Et brukstilfelle har vanligvis en hovedflyt (dvs. den mest sannsynlige) og alternative flyter.

Brukstilfelle beskriver prosessflyten gjennom et system basert på dens faktiske sannsynlige bruk, så testtilfeller basert på dem er mest nyttige for å avdekke feil i prosessflyten ved faktisk bruk av systemet. Brukstilfeller, ofte referert til som scenarier, er meget nyttige for å utvikle akseptansetester med deltagelse fra kunde/bruker. De er også nyttige for å avdekke integrasjonsfeil forårsaket av interaksjon og innblanding fra forskjellige komponenter, som testing av de individuelle komponentene ikke vil oppdage. En kan kombinere konstruksjon av testtilfelle fra brukstilfelle med andre spesifikasjonsbaserte testteknikker.

4.4 Strukturbaserte (hvit boks) teknikker (K4)*60 minutter***Begreper**

Kodedekning, beslutnings-/forgreningsdekning, programinstruksjonsdekning, strukturbasert testing.

Vi bruker i tillegg til de norske de tilsvarende engelske begrepene statement coverage og branch/decision coverage.

Bakgrunn

Strukturbasert testing (hvitboks testing) er basert på en identifisert struktur i programvaren eller systemet, slik som i følgende eksempler:

- Komponentnivå: Strukturen i koden selv, f. eks programinstruksjoner, beslutninger eller forgreninger eller til og med bestemte stier gjennom programmet.
- Integrasjonsnivå: Strukturen kan være et kallhierarki (et diagram der moduler kaller opp andre moduler).
- Systemnivå: Strukturen kan være en menystruktur, foretningsprosess eller nettsidestruktur.

I dette avsnitt diskuteres to koderelaterte strukturbaserte teknikker for kodedekning, basert på programinstruksjoner og beslutninger. For beslutningstesting kan man benytte et kontrollflytdiagram for å visualisere alternativene for hver beslutning.

4.4.1 Programinstruksjonstesting og dekning (statement testing and coverage) (K3)

I modultesting er programinstruksjonsdekning måling av prosenten av kjørbare programinstruksjoner som har blitt utført av en testsuite. Programinstruksjonstesting utleder testtilfeller som utfører spesifikke programinstruksjoner, normalt for å øke programinstruksjonsdekningen.

Programinstruksjonsdekning beregnes ved å dele antall utførbare instruksjoner dekket av konstruerte eller utførte testtilfelle med antallet av alle utførbare instruksjoner i koden under test.

4.4.2 Forgrenings- / Beslutningstesting og dekning (branch or decision testing and coverage) (K3)

Beslutningsdekning, det samme som forgreningstesting, er måling av prosenten av beslutningsresultat (dvs. Sant og Usant mulighetene i en IF-setning) som har blitt utført av en testsuite. Beslutningstesting utleder testtilfeller som utfører spesifikke beslutningsresultat, normalt for å øke beslutningsdekningen.

Beslutningsdekning beregnes ved å dele antall beslutningsresultater dekket av konstruerte eller utførte testtilfelle med antallet av alle beslutningsresultater i koden under test.

Beslutningstesting er en form for kontrollflyttesting siden den genererer en spesifikk kontrollflyt gjennom beslutningspunktene. Beslutningsdekning er bedre enn programinstruksjonsdekning: 100 % beslutningsdekning garanterer 100 % programinstruksjonsdekning, men ikke motsatt.

4.4.3 Andre strukturbaserte teknikker (K1)

Det finnes høyere nivåer for strukturdekning utover beslutningsdekning, f. eks betingelsesdekning (condition coverage) og sammensatt betingelsesdekning (multiple condition coverage).

Konseptet med dekningsgrad kan også benyttes på andre testnivåer (som på integrasjonsnivå) der prosenten av moduler, komponenter eller klasser som har blitt utført av en testsuite kan bli uttrykt som modul-, komponent- eller klassesdekning.

Støtteverktøy er nyttige for strukturbasert testing av koden.

4.5 Erfaringsbaserte teknikker (K2)	30 minutter
-------------------------------------	-------------

Begreper

Eksplorativ testing (uforskende testing), eng. fault attack.

Bakgrunn

Erfaringsbasert testing er når testene blir utledet basert på testernes kunnskap og intuisjon og deres erfaringer med lignende applikasjoner og teknologier. Når teknikken blir brukt i tillegg til systematiske teknikker kan den være nyttig for å identifisere spesielle tester som ikke så lett blir fanget opp av formelle teknikker, spesielt når erfaringsbasert testing blir anvendt i tillegg til mer formelle metoder. Denne teknikken kan likevel gi en meget varierende grad av effektivitet avhengig av testernes erfaringer.

En vanlig brukt erfaringsbasert teknikk er feilgjetting. Generelt forutser testere feil basert på erfaring. En strukturert tilnæringsmåte til feilgjettingsteknikken er å spesifisere en liste av mulige feil og lage testtilfeller som angriper disse feilene. Denne systematiske tilnæringsmåten kalles fault attack. Disse listene over problemer og feil kan lages basert på erfaringer, tilgjengelige feildata og fra vanlig kunnskap om hvorfor programvare feiler.

Eksplorativ testing er samtidig testdesign, testutførelse, testlogging og læring, basert på en testoppgave som inneholder testmål og som utføres i fast bestemte tidsintervaller. Denne tilnæringsmåten er mest nyttig når man har få eller mangelfulle spesifikasjoner og meget stort tidspress, eller for å supplere annen, mer formell testing. Den kan fungere som en sjekk på testprosessen, for å sikre at de mest alvorlige feil blir funnet.

4.6 Valg av testteknikker (K2)	<i>15 minutter</i>
---------------------------------------	--------------------

Begreper

Ingen spesifikke begreper.

Bakgrunn

Valg av testteknikker er avhengig av flere faktorer, inkludert typen system, pålagte standarder, kunde- eller kontraktskrav, risikonivå, typen risiko, testmål, tilgjengelig dokumentasjon, kunnskapen til testerne, tid og budsjett, utviklingslivssyklus, brukstilfelle (use case) modeller og tidligere erfaringer med feiltypene som finnes.

Noen teknikker er mer egnet i spesielle situasjoner og testnivåer; andre kan benyttes på alle testnivåer.

Testere bruker vanligvis en kombinasjon av testteknikker når de lager testtilfelle. De bruker prosessdrevne, regeldrevne og datadrevne teknikker for å sikre passende dekning av testobjektet.

Referanser

- 4.1 Craig, 2002, Hetzel, 1988, IEEE Standard 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 Testledelse (K3)**170 minutter***Læremål for testledelse*

Målene identifiserer hva du skal være i stand til etter å ha gjennomgått hver modul.

5.1 Testorganisasjon (K2)

- LO-5.1.1 Vite om viktigheten av uavhengig testing. (K1)
- LO-5.1.2 Liste opp fordelene og ulempene med uavhengig testing i en organisasjon. (K2)
- LO-5.1.3 Kjenne til de ulike typer medarbeidere en behøver for å sette sammen en testgruppe. (K1)
- LO-5.1.4 Kjenne de typiske oppgavene for en testleder og tester. (K1)

5.2 Testplanlegging og estimering (K3)

- LO-5.2.1 Kjenne de ulike nivåer og mål av testplanlegging. (K1)
- LO-5.2.2 Oppsummere formål og innhold av en testplan, testdesignspesifikasjon og testprosedyredokumenter i samsvar med 'Standard for Software Test Documentation' (IEEE 829-1998). (K2)
- LO-5.2.3 Skille mellom konseptuelt ulike teststrategier, som analytisk, modellbasert, metodisk, i samsvar med prosess/standard, dynamisk/heuristisk, konsultativ og regresjonsbekjempende. (K2)
- LO-5.2.4 Skille mellom testplanleggingen for et system og tidsplanleggingen for testutførelsen. (K2)
- LO-5.2.5 Skrive en tidsplan for testutførelse for et gitt sett med testtilfeller, ved å ta hensyn til prioritering og tekniske og logiske avhengigheter. (K3)
- LO-5.2.6 Liste testforberedelses- og utførelsesaktiviteter som bør vurderes under testplanlegging. (K1)
- LO-5.2.7 Kjenne typiske faktorer som har innflytelse på arbeidsmengden til testing. (K1)
- LO-5.2.8 Skille mellom to konseptuelt ulike estimeringsmetoder: basert på måldata og basert på ekspertviten. (K2)
- LO-5.2.9 Kjenne til/begrunne passende sluttkriterier for spesifikke testnivåer og grupper av testtilfeller (for eksempel for integrasjonstesting, akseptansetesting eller testtilfelle for brukbarhetstesting). (K2)

5.3 Overvåking og kontroll av testfremdrift (K2)

- LO-5.3.1 Kjenne til vanlige måldata brukt for å følge opp testforberedelse og utførelse. (K1)
- LO-5.3.2 Forstå og tolke måldata fra testing for testrapportering og styring av testingen (for eksempel feil funnet og rettet, og tester som gikk bra eller ikke) relatert til målsetninger og bruk. (K2)
- LO-5.3.3 Oppsummere mål med og innhold i en testsluttrapport i henhold til 'Standard for Software Test Documentation' (IEEE 829-1998). (K2)

5.4 Konfigurasjonsstyring (K2)

- LO-5.4.1 Oppsummere hvordan konfigurasjonsstyring understøtter testing. (K2)

5.5 Risiko og testing (K2)

- LO-5.5.1 Beskrive en risiko som et mulig problem som ville true oppnåelsen av en eller flere interessenters mål med prosjektet. (K2)
- LO-5.5.2 Huske at risikoer er bestemt ved sannsynlighet for en hendelse og dens følger (skade som følge av at det skjer). (K1)
- LO-5.5.3 Skille mellom prosjektrisikoen og produktrisikoen. (K2)
- LO-5.5.4 Kjenne typiske prosjektrisikoen og produktrisikoen. (K1)

LO-5.5.5 Beskrive ved hjelp av eksempler, hvordan risikoanalyse og risikostyring kan brukes under planlegging av testingen. (K2)

5.6 Hendelses- og feilstyring (K3)

LO-5.6.1 Kjenne innholdet i en hendelsesrapport i samsvar med 'Standard for Software Test Documentation' (IEEE 829-1998). (K1)

LO-5.6.2 Skrive en hendelsesrapport ved oppdagelsen av en feil under testing. (K3)

5.1 Testorganisasjon (K2)	30 minutter
---------------------------	-------------

Begreper

Tester, testleder

5.1.1 Testorganisasjon og uavhengighet (K2)

En kan øke effektiviteten med å finne feil under test og granskninger ved å bruke uavhengige testere. Ulike grader av uavhengighet er:

- Ingen uavhengige testere. Utviklere tester sin egen kode.
- Uavhengige testere innenfor utviklingsgrupper.
- Uavhengig testgruppe i organisasjonen med rapportering til prosjektledelse eller linjeledelse.
- Uavhengige testere fra forretningsorganisasjonen eller brukergruppen.
- Uavhengige testspesialister for spesifikke testmål som brukbarhetstestere, sikkerhetstestere eller sertifiseringstestere (som sertifiserer et programvareprodukt mot standarder og regler).
- Uavhengige testere som er underleverandører eller eksterne i forhold til organisasjonen.

For store, komplekse eller sikkerhetskritiske prosjekter er det vanligvis best å ha flere nivå av testing, hvor noen eller alle nivå utføres av uavhengige testere. Utviklingspersonale kan delta i testingen, spesielt på de lavere nivåene, men deres mangel på objektivitet begrenser ofte deres effektivitet. Uavhengige testere kan ha autoriteten til å kreve, og definere, testprosesser, standarder og regler. Testere bør dog ikke ta slike prosessrelaterte roller med mindre et klart mandat fra ledelsen er tilstede.

Fordelene med uavhengighet omfatter:

- Uavhengige testere ser andre og ulike feil, og er ikke forutinntatt.
- En uavhengig tester kan verifisere antagelser som folk gjorde under spesifisering og implementasjon av systemet.

Ulemper omfatter:

- Isolasjon fra utviklingsgruppen (hvis testerne blir behandlet som totalt uavhengige).
- Uavhengige testere kan bli ansett som en flaskehals dersom de er siste sjekkpunkt, og de kan bli beskyldt for å forsinke frigivelsen.
- Utviklere kan miste ansvarsfølelsen for kvaliteten.
- Uavhengige testere må lære seg systemet og/eller fagfeltet (dette punktet er ikke nevnt i den engelske versjonen av pensum).

Testoppgaver kan utføres av folk i en spesiell testrolle eller av personer i andre roller, som for eksempel prosjektleder, kvalitetsleder, utvikler, forretnings- eller anvendelsesekspert eller av folk i IT-drift eller infrastruktur.

5.1.2 Testerens og testlederens oppgaver (K1)

I dette pensum dekkes to testroller, testleder og tester. Aktivitetene og oppgavene som personer i disse to roller utfører, avhenger av prosjektets og produktets kontekst, personene i rollene og organisasjonen.

Noen ganger blir testlederen kalt testmanager eller testkoordinator. Testlederrollen kan utføres av en prosjektleder, utviklingsleder, kvalitetsleder eller lederen for en testgruppe. I store prosjekter kan

testlederrollen være oppdelt i flere nivåer. Typisk vil en testleder planlegge, følge opp og styre testaktivitetene som er definert i avsnitt 1.4.

Typiske testlederoppgaver kan omfatte:

- Koordinere teststrategi og plan med prosjektledere og andre.
- Skrive eller gjennomgå en teststrategi for prosjektet, og testpolicy for organisasjonen.
- Bidra med testperspektivet til andre prosjektaktiviteter, som integrasjonsplanlegging.
- Planlegge testene – ved å ta hensyn til kontekst, testmålsetninger og risikoer – inklusive å velge teststrategier, å estimere tiden, arbeidsmengden og kostnaden av testing, å få tak i ressurser, å definere testnivåer og testsykluser, og å planlegge hendeshåndteringen.
- Starte opp spesifisering, forberedelse, implementering og utføringen av tester, overvåke testresultater og sjekke sluttkriterier.
- Tilpasse planleggingen på basis av testresultater og framdrift (noen ganger dokumentert i statusrapporter) og foreta enhver handling som behøves for å motvirke problemer.
- Sette opp passende konfigurasjonsstyring av testmateriale for sporbarhet.
- Innføre passende måledata for å måle testframdrift og for å evaluere kvaliteten av testingen og produktet.
- Beslutte hva som skal bli automatisert, i hvilken utstrekning og hvordan.
- Velge ut verktøy for å støtte testingen og organisere enhver opplæring i bruk av verktøy for testerne.
- Ta beslutninger omkring implementeringen av testmiljøet.
- Skrive sluttrapporter fra testingen basert på informasjon som er samlet under testing.

Typiske oppgaver for en tester kan omfatte:

- Gjennomgå og bidra til testplaner.
- Analysere, gjennomgå og bedømme brukerkrav, spesifikasjoner og modeller vedrørende testbarhet.
- Lage testspesifikasjoner.
- Sette opp testmiljø (ofte koordinert med systemforvaltning og nettverksadministrasjon).
- Forberede og fremskaffe testdata.
- Implementere tester på alle nivåer, utføre og logge disse, evaluere resultater og dokumentere avvik fra forventede resultater.
- Bruke verktøy for testadministrasjon eller testledelse samt verktøy for oppfølging av testingen som påkrevd.
- Automatisere tester (dette kan være støttet av en utvikler eller en automatiseringsekspert).
- Måle ytelse av komponenter og system (hvor det passer).
- Gjennomgå tester som er utviklet av andre.

Personer som arbeider med testanalyse, testdesign, spesifikke testtyper eller testautomatisering kan være spesialisert i disse rollene. Avhengig av testnivå og risiko ved produkt og prosjekt kan ulike personer overta rollen som testere, med ulike grader av uavhengighet. Typiske testere på komponent- og integrasjonsnivå er utviklere, testere på akseptansetestnivå er eksperter på anvendelsesområde og brukere, og testere for driftsakseptansetesting er operatører.

5.2 Test planlegging og estimering (K3)**40 minutter****Begreper**

Testmetode, teststrategi

5.2.1 Testplanlegging (K2)

Dette avsnitt dekker testplanleggingens formål i utviklings- og implementeringsprosjekter, og for vedlikeholdsaktiviteter. Planlegging kan dokumenteres i en overordnet testplan, og i separate testplaner for de enkelte testnivåene som systemtesting eller akseptansetesting. Maler til innhold i testplaner er gitt i 'Standard for Software Test Documentation' (IEEE 829-1998).

En organisasjons testpolitikk, testingens omfang, mål, risikoer, begrensninger, kritikalitet, testbarhet og tilgjengelighet til ressurser har innflytelse på dens testplanlegging. Jo mer prosjektet og testplanleggingen skrider fram, desto mer informasjon er tilgjengelig og desto mer detaljer kan inkluderes i planen.

Testplanlegging er en kontinuerlig aktivitet og blir utført i alle livssyklusprosesser og aktiviteter. Tilbakemelding fra testaktiviteter blir brukt for å oppfatte endringer i risikobildet slik at planleggingen kan tilpasses.

5.2.2 Testplanleggingsaktiviteter (K2)

Testplanlegging for et helt system eller et delsystem kan omfatte:

- Å bestemme omfang og risiko, og identifisere målene for testingen.
- Å definere den totale strategien eller metoden for testingen, inklusive definisjonen av testnivåer, samt start- og sluttkriterier.
- Å integrere og koordinere testaktivitetene med livssyklusaktivitetene til programvaren: Kjøp, leveranse, utvikling, drift og vedlikehold.
- Å beslutte hva en skal teste, hvilke roller som skal utføre testaktivitetene, hvordan disse bør utføres og hvordan en skal evaluere testresultater.
- Å planlegge tidspunkt for testanalyse og designaktiviteter.
- Å planlegge tidspunkter for testimplementering, utførelse og evaluering.
- Å tildele ressurser til de ulike definerte aktivitetene.
- Å definere omfang, detaljnivå, struktur og maler for testdokumentasjon.
- Å velge måldata for å følge opp og styre testforberedelse og utførelse, feilretting og risiko.
- Å sette detaljnivået for testprosedyrer for å gi nok informasjon til å understøtte reproduserbar testforberedelse og utføring.

5.2.3 Startkriterier (K2)

Startkriterier definerer når en kan starte testingen, som for eksempel et testnivå eller når en samling tester er klar for utføring. Typisk kan startkriterier dekke følgende:

- At testmiljø er klart og tilgjengelig
- At testverktøy er klare i testmiljøet
- At testbar kode er tilgjengelig
- At testdata er tilgjengelige

5.2.4 Sluttkriterier (K2)

Hensikten med sluttkriterier er å definere når en kan avslutte testingen, som for eksempel på slutten av et testnivå eller når en samling tester har oppnådd et spesifikt mål.

Typiske sluttkriterier kan bestå av:

- Grundighetsmål, som kodedekning, funksjonsdekning eller risiko.
- Estimater av feiltetthet eller pålitelighetsmål.
- Kostnad.
- Gjenværende risikoer, som ikke rettede feil eller mangel på testdekning i visse områder.
- Tidsplaner som er basert på produksjonssettingsdato.

5.2.5 Testestimering (K2)

To metoder for estimering av testarbeidet er dekket i pensum:

- Basert på måledata: Å estimere testarbeidet på grunnlag av måledata fra tidligere eller liknende prosjekter eller basert på typiske verdier,
- Ekspertbasert: Å estimere oppgavene ved at den som skal ha ansvaret for oppgavene eller eksperter estimerer.

Når testarbeidet er estimert kan en identifisere ressurser og lage en tidsplan.

Testarbeidets omfang kan avhenge av en rekke faktorer, inklusive:

- Produktegenskaper: Kvaliteten til spesifikasjonen og annen informasjon som brukes for testmodellene (dvs. testgrunnlaget), produktets størrelse, problemområdets kompleksitet, kravene til pålitelighet og sikkerhet, og krav til dokumentasjon.
- Utviklingsprosessens egenskaper: Organisasjonens stabilitet, verktøy som blir brukt, testprosessen, medarbeidernes ferdigheter, og tidspress.
- Testresultatet: Antall feil og omfang av gjentatt arbeid som kreves.

5.2.6 Teststrategier og testmetoder (K2)

Testmetoden er realiseringen av teststrategien for et spesifikt prosjekt. Testmetoden blir definert og detaljert i testplanene og testdesign. Typisk inneholder den beslutningene som er tatt på grunnlag av prosjektets mål og risikobedømmelse. Den er utgangspunkt for å planlegge testprosessen, for å velge teknikker for testdesign og testtyper som skal brukes og for å definere start- og sluttkriterier.

Metoden som velges avhenger av sammenhengen og en kan ta hensyn til risiko, farer og sikkerhet, tilgjengelige ressurser og evner, teknologi, systemets natur (for eksempel spesielt bygget eller standardssystem), testmål og lover og forskrifter.

Typiske metoder omfatter følgende metoder:

- Analytiske metoder, som risikobasert testing der testing blir rettet mot områdene med størst risiko.
- Modellbaserte metoder, som for eksempel tilfeldig stikkprøvetesting ved bruk av statistisk informasjon om feilrater (som pålitelighetsmodeller) eller bruk (som bruksprofiler).
- Metodiske metoder, som feilbasert (inkludert feilgjetting og angrep basert på feil), erfaringsbasert, sjekklisterbasert og kvalitetsegenskapsbasert.
- Prosess- eller standardkonforme metoder, som de som er spesifisert i bransjestandarder.

- Dynamiske eller heuristiske metoder, som eksplorativ testing (uforskende testing) eller ulike agile metoder der testing er mer reaktiv til hendelser enn planlagt på forhånd, og der utføring og evaluering skjer samtidig.
- Konsultative metoder, som slike der testdekning er fortrinnsvis drevet av råd og veiledning fra teknologiske eksperter og/eller forretningseksperter utenfor testgruppen.
- Regresjonsbekjempende metoder, som de som omfatter gjenbruk av eksisterende testmateriale, omfattende automatisering av funksjonelle regresjonstester, og standard testsuiter.

Ulike metoder kan kombineres, for eksempel, en risikobasert dynamisk metode.

5.3 Overvåking og kontroll av testframdriften (K2)

20 minutter

Begreper

Feiltetthet, feilrate, teststyring, testoppfølging, testrapport

5.3.1 Oppfølging av testframdrift (K1)

Formålet med å følge opp testen er å gi tilbakemelding og synlighet omkring testaktiviteter. Informasjonen som skal følges opp kan samles manuelt eller automatisk og kan brukes for å måle sluttkriterier som for eksempel testdekning. Måledata og metrikker kan også brukes til å bedømme framdrift i forhold til tidsplan og budsjett. Vanlige måledata omfatter:

- Prosent arbeid utført i forberedelse av testtilfelle (eller prosent av planlagte testtilfelle forberedt).
- Prosent arbeid utført i forberedelse av testmiljø.
- Utføring av testtilfelle (for eksempel antall testtilfelle utført eller ikke og antall testtilfelle kjørt med rett eller feil resultat).
- Informasjon om feil (for eksempel feiltetthet, feil funnet og rettet, feilrate og resultatene av fornyet test).
- Testdekning mot krav eller andre parameter.
- Testernes subjektive tillit til produktet.
- Dato for testmilepæler.
- Testkostnader, inklusive kostnad sammenlignet med nytteverdien av å finne den neste feilen eller å kjøre den neste testen.

5.3.2 Testrapportering (K2)

Testrapportering dreier seg om å oppsummere informasjon om testingen, inklusive:

- Hva skjedde under testingen, som dato når sluttkriterier ble oppfylt.
- Analysert informasjon og måledata for å støtte anbefalinger og beslutninger om framtidige handlinger, som å bedømme feil som er igjen, den økonomiske nytten av fortsatt testing, fortsatt eksisterende risikoer, og tiltroen til den testede programvaren.

Innholdsfortegnelsen til en sluttrapport fra testingen er gitt i 'Standard for Software Test Documentation' (IEEE 829-1998).

Måledata bør samles under testingen og ved slutten av et testnivå for å bedømme:

- Om testmålene til testnivået var passende.
- Om tilnærmingen til testen var passende.
- Effektiviteten av testingen mht. dens mål.

5.3.3 Teststyring (K2)

Teststyring beskriver enhver beslutning eller endring gjort som et resultat av informasjon og måledata som er samlet og rapportert. Handlingene kan gjelde enhver testaktivitet og kan ha innflytelse på enhver annen livssyklusaktivitet eller oppgave for programvaren.

Eksempler for teststyringshandling er:

- Å ta beslutninger basert på informasjon fra testoppfølgingen.
- Å omprioritere tester når en identifisert risiko oppstår (for eksempel at programvaren blir levert forsinket).
- Å endre tidsplanen for testingen på grunn av (u)tilgjengeligheten til et testmiljø.

- Å sette et startkriterium som krever at feilrettinger har blitt testet om igjen av en utvikler før de blir akseptert i en build (bygg).

5.4 Konfigurasjonsstyring (K2)	10 minutter
---------------------------------------	--------------------

Begreper

Konfigurasjonsstyring, versjonskontroll

Bakgrunn

Målet med konfigurasjonsstyring er å etablere og vedlikeholde integriteten til produktene (komponentene, data og dokumentasjon) som tilhører programvaren eller systemet gjennom prosjektets eller produktets livssyklus.

For testing kan konfigurasjonsstyring bety at:

- alle deler av testmateriale er identifisert og underlagt versjonskontroll, at endringer er styrt, at det finnes informasjon om hvordan delene hører til hverandre og til det som er utviklet (testobjektene), slik at sporbarhet kan opprettholdes gjennom testprosessen.
- alle identifiserte dokumenter og utviklingsprodukter er entydig identifisert i testdokumentasjonen.

For testeren hjelper konfigurasjonsstyring å entydig identifisere og reprodusere hva som er testet, testdokumentene, testene og testrammen(e).

Prosedyrer og infrastruktur (verktøy) for konfigurasjonsstyring bør velges, dokumenteres og implementeres under testplanleggingen.

5.5 Risiko og testing (K2)	30 minutter
-----------------------------------	--------------------

Begreper

Produktrisiko, prosjektrisiko, risiko, risikobasert testing

Bakgrunn

Risiko kan defineres som muligheten for at en hendelse, fare, trussel eller situasjon vil inntre samt dens uønskede konsekvenser, altså et mulig problem. Risikonivået bestemmes av sannsynligheten for at en negativ hendelse inntre og følgen av denne.

5.5.1 Prosjektrisikoen (K2)

Prosjektrisikoen er de risikoelementene som kan påvirke et prosjekts evne til å levere dets mål, som for eksempel:

- Organisatoriske faktorer:
 - Knapphet på kunnskap, opplæring, personale og evner;
 - Problemer med personale;
 - Politiske problemer som for eksempel
 - Problemer for testere å formidle deres behov og testenens resultater;
 - Ingen oppfølging av informasjon funnet under test og granskninger (for eksempel ingen forbedring av utviklings- eller testpraksis).
 - Dårlig holdning eller feil forventning til testing (for eksempel å ikke se verdien av at feil blir funnet under testing).
- Tekniske faktorer:
 - Problemer med å definere de riktige krav;
 - Hvor vidt kravene ikke kan oppfylles under gitte omstendigheter og restriksjoner;
 - Testomgivelsen er ikke ferdig tidlig nok;
 - For sen datakonvertering, planlegging av migrering og verktøy for å konvertere eller migrere data til utvikling og test;
 - Kvaliteten av design, kode, testdata og testtilfeller.
- Leverandørfaktorer:
 - Problemer hos en tredje part (underleverandør);
 - Problemer med kontrakter.

Ved analyse, styring og behandling av slike risikoer følger testlederen veletablerte prinsipper for prosjektledelse. Standarden IEEE 829-1998 'Standard for Software Test Documentation' krever at risikoer og aksjoner for å behandle dem blir nevnt.

5.5.2 Produktrisikoen (K2)

Potensielle områder som kan feile (negative hendelser i framtiden eller farer) i programvaren eller systemet er kjent som produktrisikoen, fordi de er en risiko til produktkvaliteten, som for eksempel:

- Feilbefengt programvare som er levert.
- Muligheten for at programvaren/plattformen kan forvolde skade på person eller bedrift.
- Dårlige programvareegenskaper (som funksjonalitet, pålitelighet, brukbarhet og ytelse).
- Dårlig dataintegritet og kvalitet (for eksempel problemer med datamigrering, konvertering, transport, avvik fra datastandarder)
- Programvare som ikke utfører de ønskede funksjoner.

Risikoer blir brukt for å beslutte hvor en skal starte testingen og hvor en skal teste mer eller mindre. Testingen brukes til å redusere risikoen for at en negativ hendelse skjer, eller til å redusere effekten av en slik hendelse.

Produktrisikoen er en spesiell type risiko for prosjektets suksess. Testing som risikostyring gir tilbakemelding på den resterende risikoen ved å måle effekten av at kritiske feil fjernes og av alternative planer som er brukt.

En risikobasert tilnærming til testing gir proaktive muligheter til å redusere nivået av produktrisiko, og den kan begynne ved prosjektoppstart. Den omfatter identifikasjon av produktrisikoen til bruk for å lede planlegging og styring, spesifisering, forberedelse og utføring av testene. Ved en risikobasert tilnærming kan de identifiserte risikoer brukes til å:

- Bestemme testteknikkene som skal brukes.
- Bestemme hvor mye test som skal utføres.
- Prioritere testingen i et forsøk på å finne de kritiske feil så tidlig som mulig.
- Bestemme om aktiviteter som ikke er del av testingen kan brukes til å redusere risikoen (for eksempel å sørge for opplæring til uerfarne utviklere).

Risikobasert testing trekker på den kollektive kunnskapen til de involverte i et prosjekt for å bestemme risikoer og hvor mye testing som kreves for å møte disse risikoer.

For å sikre at sannsynligheten for at prosjektet mislykkes blir minimert, gir risikostyringsaktiviteter en disiplinert tilnærming til å:

- bedømme (og bedømme om igjen regelmessig) hva som kan gå galt (risiko).
- bestemme hvilke risikoer som er viktige å håndtere.
- gjennomføre tiltak for å håndtere disse risikoer.

I tillegg kan testing hjelpe med å identifisere nye risikoer, hvilke risikoer som burde reduseres, og redusere usikkerheten omkring risikoer.

5.6 Hendelses- og feilhåndtering (K3)

40 minutter

Begreper

Hendelsesregistrering, feilregistrering, feilhåndtering, hendelseshåndtering, hendelsesrapport

Bakgrunn

For di et av målene med testingen er å finne feil, må avvikene mellom virkelige og forventede utdata og tilstander registreres som hendelser. En hendelse må undersøkes og kan da vise seg å være en feil. Passende tiltak for å behandle hendelser og feil må defineres. Hendelser og feil må følges fra de blir oppdaget og klassifisert til de er korrigert og løsningen er kontrollert og funnet rett. For å kunne håndtere alle hendelser til de er avsluttet bør en organisasjon etablere en prosess for hendelseshåndtering og regler for klassifisering.

Hendelser kan registreres under utvikling, granskning, testing eller bruk av programvareproduktet. De kan registreres for saker i koden eller systemet, eller i enhver type dokumentasjon som omfatter krav, utviklingsdokumenter, testdokumenter og brukerinformasjon som "hjelp" eller installasjonsveiledning.

Hendelsesrapporter har følgende mål:

- Å gi tilbakemelding til utviklere og andre involverte om et problem for å muliggjøre identifikasjon, isolering og retting som nødvendig.
- Å gi testledere en mulighet til å følge opp kvaliteten til systemet under test og testingens framskritt.
- Å gi ideer til forbedring av testprosessen.

Bemerkning (bare i norsk versjon av denne pensumlisten): Ordet "hendelse" brukes for å betegne en sak som behøver undersøkelse, men der en ikke er sikker på om det er en feil. Det kan også være et problem i f.eks. testmaterialet.

Detaljer i en hendelsesrapport kan omfatte:

- Dato, organisasjon og forfatter som skriver rapporten.
- Forventede og virkelige resultater.
- Identifikasjon av testobjektet (konfigurasjonselementet) og omgivelsen.
- Software- eller systemlivssyklusprosess som hendelsen ble observert i.
- Beskrivelsen av hendelsen for å muliggjøre gjenskapelse og retting, noe som kan inkludere logger, kopier av databasen eller kopier av skjermbilder.
- Omfang eller virkning på den rapporterende parts eller andres interesser.
- Alvorligheten av feilens følger for systemet.
- Hvor mye det haster med rettelsen.
- Status av hendelsen (for eksempel åpen, utsatt, duplisert, venter på rettelse, rettet men ikke re-testet, lukket).
- Slutninger, anbefalinger og godkjenninger.
- Globale forhold, som for eksempel at endringen som kommer fra en hendelse kan ha virkninger på andre områder.
- Endringshistorie, som viser rekkefølgen av handlingene foretatt av prosjektgruppens medlemmer med hensyn til hendelsen for å isolere, rette og sjekke den, og vise at den er rettet.
- Referanser, inklusive identifikasjonen av testtilfelle(ne) som avslørte problemet.

Strukturen til en hendelsesrapport er også dekket i 'Standard for Software Test Documentation' (IEEE 829-1998).

Referanser

- 5.1.1 Black, 2001, Hetzel, 1988
- 5.1.2 Black, 2001, Hetzel, 1988
- 5.2.5 Black, 2001, Craig, 2002, IEEE 829-1998, 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 Verktøystøtte til test (K2)	80 minutter
--------------------------------------	--------------------

Læremål for verktøystøtte til test

Læremålene viser hva du skal være i stand til etter fullførelsen av hver modul.

6.1 Typen testverktøy (K2)

- LO-6.1.1 Klassifisere forskjellige typer testverktøy i henhold til deres oppgaver og aktivitetene i den fundamentale testprosessen og i programvarens livssyklus. (K2)
- LO-6.1.2 Forklare termen testverktøy og formålet med verktøystøtte i testing. (K2)

6.2 Effektiv bruk av verktøy: mulige fordeler og risikoer (K2)

- LO-6.2.1 Oppsummere mulige nytteeffekter og risikoer ved testautomatisering og verktøystøtte for testing. (K2)
- LO-6.2.2 Huske spesielle vurderinger for verktøy for testutføring, statistisk analyse, testadministrasjon og testledelse. (K1)

6.3 Introduksjon av et verktøy i en organisasjon (K1)

- LO-6.3.1 Oppgi hovedprinsippene for å introdusere et verktøy i en organisasjon. (K1)
- LO-6.3.2 Oppgi målene for proof-of-concept for verktøyevaluering, og for pilotfasen for å implementere/driftsette verktøyet. (K1)
- LO-6.3.3 Vite at det kreves mer enn anskaffelse av et verktøy for å oppnå god verktøystøtte. (K1)

6.1 <i>Typer testverktøy (K2)</i>	45 minutter
--	--------------------

Begreper

Konfigurasjonsstyringsverktøy, dekningsverktøy, debuggings-/avlusningsverktøy, dynamisk analyseverktøy, hendelses-/feilhåndteringsverktøy, lasttestverktøy, modelleringsverktøy, overvåkningsverktøy, ytelsestestverktøy, probeeffekt/instrumenteringseffekt/verktøyeffekt, kravhåndteringsverktøy, granskningsverktøy, sikkerhetsverktøy, statistisk analyseverktøy, stresstestingsverktøy, sammenligningsverktøy, testdata forberedelsesverktøy, testdesignverktøy, testramme (stubber, drivere), testutføringsverktøy, testadministrasjonsverktøy, testrammeverktøy for komponenttest.

6.1.1 Forstå mål og mening med verktøystøtte for testing (K2)

Testverktøy kan benyttes i flere testrelaterte aktiviteter. Slike verktøy kan være:

1. Verktøy som direkte benyttes til testing, som testutføringsverktøy, verktøy for å generere testdata og sammenligningsverktøy.
2. Verktøy som hjelper til å administrere testprosessen, slik som å håndtere tester, testresultater, data, krav, hendelser, defekter etc. og for å rapportere og overvåke testutføringen.
3. Verktøy som er brukt i rekognosering, eller utforskning og leting (for eksempel verktøy som overvåker filaktiviteten til en applikasjon).
4. Ethvert verktøy som er et hjelpemiddel i testingen (et regneark er også et testverktøy i denne sammenheng).

Verktøystøtte til testing kan ha ett eller flere formål avhengig av konteksten:

- Forbedre effektiviteten av testaktivitetene ved å automatisere gjentakende oppgaver eller å støtte manuelle testaktiviteter som testplanlegging, testdesign, testrapportering og overvåking.
- Automatisere aktiviteter som krever vesentlige ressurser når de utføres manuelt (for eksempel statistisk testing).
- Automatisere aktiviteter som ikke kan utføres manuelt (for eksempel storskala ytelsestesting av klient-tjener applikasjoner).
- Øke pålitelighet av testingen (for eksempel ved å automatisere store datasammenligninger eller å simulere en oppførsel).

For dette pensumet benyttes "testrammeverk" iht. punktene "Testutføringsverktøy" og "Testrammeverktøy/Rammeverk for enhetstest (U)" som beskrevet i seksjon 6.1.6.

For øvrig bruker industrien termen "testrammeverk" også til minst tre andre formål:

- Gjenbrukbare og utvidbare testbibliotek som kan bli benyttet til å bygge testverktøy (også kalt testramme)
- En type av design av automatiserte tester (for eksempel datadrevet, nøkkelorddrevet)
- Samlet prosess for utføring av tester

6.1.2 Klassifisering av testverktøy (K2)

Det finnes en rekke verktøy som støtter forskjellige aspekter ved testing. Verktøy kan klassifiseres basert på forskjellige kriterier som formål, lisensiering (kommersiell / gratis / open-source / shareware), brukt teknologi osv. I dette pensumet er verktøy klassifisert i henhold til de testaktiviteter verktøyene støtter.

Noen verktøy støtter tydelig én bestemt aktivitet, andre kan støtte flere aktiviteter. De blir klassifisert under den aktivitet de er nærmest assosiert med. Verktøy fra en enkel leverandør, spesielt de verktøy som er designet til å fungere sammen, kan bli samlet i en felles verktøypakke/suite.

Noen typer verktøy kan være invaderende, eller påtrengende, ved at verktøyet selv har effekt på testresultatet. For eksempel kan utførelsetidspunkter være forskjellig pga. ekstra kode som utføres av verktøyet, eller en kan få ulike verdier for kodedekning avhengig av hva slags verktøy for dekningsmåling en bruker. Konsekvensen av påtrengende verktøy kalles verktøyeffekten eller instrumenteringseffekten.

Noen verktøy tilbyr støtte mer tilpasset utviklere (for eksempel ved komponent og komponent integrasjons testing). Slike verktøy er merket med "(U)" i listen under.

6.1.3 Verktøystøtte for administrasjon av testing og tester (K1)

Administrasjonsverktøy støtter alle testaktiviteter gjennom hele programvarens livssyklus.

Testadministrasjonsverktøy

Disse verktøyene tilbyr grensesnitt for å utføre tester, spore feil og håndtere krav samtidig med å støtte kvantitativ analyse og rapportering av testobjektene. De støtter også det å spore testobjektene til kravspesifikasjonene og kan ha uavhengig mulighet for versjonskontroll eller et grensesnitt til et eksternt verktøy for versjonskontroll.

Kravhåndteringsverktøy

Disse verktøyene lagrer kravbeskrivelser, lagrer attributter til krav (inklusive prioritet), tilbyr unike identifikatorer og støtter sporing av krav til individuelle tester. Disse verktøyene kan også hjelpe til å identifisere inkonsistente eller manglende krav.

Feil-/Hendeshåndteringsverktøy

Disse verktøyene lagrer og håndterer avviks-/hendelsesrapporter, dvs. defekter, feil, endringsønsker eller antatte problemer og anomalier/avvik, og hjelper til i å håndtere livssyklusen til hendelsene, med mulig støtte til statistisk analyse.

Konfigurasjonsstyringsverktøy

Selv om disse ikke er direkte testverktøy så er disse verktøyene nødvendig for å lagre og styre versjonering av testware og relatert programvare spesielt ved konfigurering av mer enn ett maskinvare/programvare miljø i form av versjon av operativsystem, kompilatorer, weblesere, etc.

6.1.4 Verktøystøtte for statisk testing (K1)

Verktøy for statisk testing tilbyr en kostnadseffektiv måte å finne flere defekter på i en tidlig fase i utviklingsprosessen.

Granskningsverktøy

Disse verktøyene gir støtte til granskningsprosessen, sjekklister, guider for granskningen og er brukt til å lagre og kommunisere granskningskommentarer, rapportere avvik og innsats/forbruk. De kan i tillegg hjelpe til å støtte online granskninger for store eller geografisk atskilte team.

Statiske analyseverktøy (U)

Disse verktøyene hjelper utviklere og testere til å finne defekter før dynamisk testing ved å tilby støtte til å håndheve kodenstandarder (inkludert sikker koding), analyse av strukturer og avhengigheter. De kan også hjelpe i planlegging eller risikoanalyse ved å tilby metrikker for koden (for eksempel kompleksitet).

Modelleringsverktøy (U)

Disse verktøyene er brukt til å validere programvaremodeller (for eksempel fysiske datamodeller (PDM) for en relasjonsdatabase), ved å liste opp inkonsistenser og å finne defekter. Disse verktøyene kan ofte hjelpe til å generere testtilfelle basert på modellen.

6.1.5 Verktøystøtte for testspesifikasjon (K1)

Testdesignverktøy

Disse verktøyene brukes til å generere inputdata til tester eller eksekverbare tester og/eller testorakler fra krav, grafiske brukergrensesnitt, designmodeller (tilstand, data eller objekt) eller kode.

Verktøy som hjelper med forberedelse av testdata

Slike verktøy manipulerer databaser, filer eller dataoverføringer for å sette opp testdata til bruk i utføring av tester for blant annet å verne om sikkerhet gjennom anonymiserte data.

6.1.6 Verktøystøtte for testutføring og logging (K1)

Testutføringsverktøy

Testutføringsverktøy gjør det mulig å utføre tester helt eller delvis automatisk ved bruk av lagrede inputdata og forventede resultater med et skriptspråk og tilbyr vanligvis en testlogg for hver testkjøring. De kan også benyttes til å gjøre opptak av en test og tilbyr normalt et skriptspråk eller en GUI-basert konfigurering og parameterisering av data og annen tilpasning av testene.

Testrammeverktøy/Rammeverk for enhetstest (U)

En testramme eller testrammeverk forenkler testing av komponenter eller deler av et system ved å simulere miljøet som objektet kjører i med bruk av "mock"-objekter som stubber og drivere.

Sammenligningsverktøy

Slike verktøy viser forskjell mellom filer, databaser eller testresultater. Testutføringsverktøy omfatter typisk dynamisk sammenligning, men sammenligning etter testutføring kan også bli utført av et eget verktøy. Et sammenligningsverktøy kan bruke et testorakel, spesielt om det er automatisert.

Dekningsmålingsverktøy (U)

Dekningsmålingsverktøy kan med invaderende eller ikke-invaderende teknikker måle prosentdelen av spesifikke typer av kodelinjer som har blitt utført (kodelinjer, beslutninger eller forgreninger og modul- eller funksjonskall) ved hjelp av et sett av tester.

Verktøy for sikkerhetstesting

Disse verktøyene brukes til å evaluere sikkerhetsegenskaper i programvare. Dette omfatter evaluering av programvarens evne til å beskytte datakonfidensialitet, integritet, autentisering, autorisasjon, tilgjengelighet og ikke-fornektning (non-repudiation). Verktøy for sikkerhetstester er ofte fokusert på bestemte teknologier, plattformer og formål.

6.1.7 Verktøystøtte til ytelse og overvåkning (K1)

Dynamiske analyseverktøy (U)

Dynamiske analyseverktøy finner feil, som bare åpenbares når programvaren utføres, slik som tidsavhengigheter og minnelekkasjer. De benyttes typisk i komponent- og komponentintegrasjonstest og ved test av mellomvare.

Ytelses- / last- / stresstestverktøy

Ytelsestestverktøy overvåker og rapporterer hvordan et system oppfører seg under varierte simulerte brukssituasjoner i form av antall samtidige brukere, deres ramp-up (økning i antall brukere) mønster, frekvens og relativ prosentfordeling av transaksjoner. Simulering av last oppnås ved å anvende virtuelle brukere som utfører utvalgte transaksjoner, spredd på forskjellige testmaskiner normalt kjent som lastgeneratorer.

Overvåkningsverktøy

Overvåkningsverktøy analyserer, verifiserer og rapporterer kontinuerlig bruken av spesifikke systemressurser, og gir advarsel ved mulige ressurs- eller tjenesteproblemer.

6.1.8 Verktøystøtte for spesifikke testbehov (K1)

Bedømmelse / vurdering av datakvalitet

Data er i sentrum for enkelte prosjekter slik som datakonverterings- og migreringsprosjekter og i applikasjoner som datavarehus. Dataenes attributter kan variere i form av kritikalitet og volum. I en slik sammenheng trenger man verktøy for å bedømme datakvalitet for å granske og verifisere regler for datakonvertering og migrering for å sikre at de prosesserte data er korrekte, komplette og samsvarer med en forhåndsdefinert kontekstspesifikk standard.

Det brukes andre verktøy for testing av brukbarhet.

<p>6.2 Effektiv bruk av verktøy: mulige fordeler og risikoer (K2)</p>	<p>20 minutter</p>
---	--------------------

Begreper

Datadrevet testing, nøkkelorddrevet testing, skriptspråk.

6.2.1 Mulige fordeler og risikoer ved verktøystøtte til testing (for alle verktøy) (K2)

Bare det å kjøpe eller leie et verktøy garanterer ikke suksess ved bruk av verktøyet. Hver type verktøy kan kreve ytterligere innsats for å oppnå reelle og varige fordeler. Det er potensielle fordeler og muligheter ved bruk av verktøy i testing, men det er også risikoer.

Potensielle fordeler ved å benytte et verktøy omfatter:

- Repeterende arbeid reduseres (f.eks. å kjøre regresjonstest, oppgi de samme testdata på nytt og sjekke mot kodenstandarder).
- Bedre konsistens og repeterbarhet (f.eks. test utført med et verktøy i samme rekkefølge og frekvens, og tester utledet fra krav).
- Objektiv vurdering (f.eks. statistiske målinger, dekning)
- Enkel tilgang til informasjon om tester eller testingen (f.eks. statistikk og grafer om testframdrift, hendelsesfrekvens og ytelse).

Risikoer ved bruk av verktøy omfatter:

- Urealistiske forventninger til verktøyet (inkludert funksjonalitet og hvor lett det er å bruke)
- Undervurdering av tid, kostnader og innsats til introduksjon av et verktøy (inkludert opplæring og ekstern ekspertise).
- Undervurdering av tid og innsats som kreves for å oppnå betydelige og kontinuerlige fordeler fra verktøyet (inkludert behov for endringer i testprosessen og kontinuerlig forbedring i hvordan verktøyet benyttes).
- Undervurdering av innsatsen som kreves for å vedlikeholde testartefaktene produsert av verktøyet.
- For stor tillit til verktøyet (erstatning for testdesign eller bruk av automatiserte tester der manuelle tester ville være bedre).
- Neglisjering av versjonskontroll av testartefakter i og generert av verktøyet.
- Neglisjering av problemer med relasjoner og interoperabilitet mellom kritiske verktøy, som kravhåndteringsverktøy, versjonskontroll, feilhåndteringsverktøy, feilforvaltningsverktøy og verktøy fra forskjellige leverandører.
- Risiko for at leverandør går konkurs, slutter med verktøyet eller selger verktøyet til en annen leverandør.
- Dårlig respons fra leverandør på support/støtte, oppgraderinger og feilrettinger.
- Uforutsette risikoer, som at det ikke støtter nye plattformer.
- Risiko for at open-source eller freeware-prosjekter avsluttes.

6.2.2 Spesielle hensyn for noen typer verktøy (K1)

Testutføringsverktøy

Testutføringsverktøy utfører tester på testobjekter ved å bruke automatiserte skript. Denne type verktøy krever ofte en stor innsats for å få vesentlig nytte.

Å lage tester ved å gjøre opptak av manuelle tester virker attraktivt, men er en framgangsmåte som ikke skalerer til et stort antall automatiserte testskript. Et testopptak er en lineær representasjon

med spesifikke data og aksjoner som en del av skriptet. Denne type skript kan bli ustabil når uventede hendelse inntreffer.

En datadrevet framgangsmåte separerer ut inputdata, vanligvis i et regneark, og bruker et mer generisk testskript som kan lese inputdata og utføre det samme testskriptet med forskjellige data. Testere som ikke er kjent med skriptspråket kan da lage testdata for disse predefinerte skriptene.

Det finnes andre teknikker som kan benyttes i datadrevet teknikk. En mulighet er hardkodete datakombinasjoner i et regneark. En annen mulighet er generering av data ved hjelp av algoritmer basert på konfigurerbare parametere som ved utføring gis som input til applikasjonen. For eksempel kan et verktøy bruke en algoritme som genererer en tilfeldig bruker-Id, og for å få repeterbarhet i et mønster, blir en startverdi (seed) benyttet for å kontrollere tilfeldighet.

I en nøkkelorddrevet framgangsmåte inneholder regnearket nøkkelord som beskriver aksjonene som skal utføres (også kalt aksjonsord – engelsk "action words") og testdata. Testere (også om de ikke er kjent med skriptspråket) kan da definere tester ved bruk av nøkkelord, som kan skreddersys til applikasjonen som skal testes.

Teknisk ekspertise i skriptspråket er nødvendig for alle framgangsmåter (enten av testere eller av spesialister i testautomatisering).

Uavhengig av skriptteknikk som benyttes, så må det forventede resultatet for hver test lagres for senere sammenligning.

Statiske analyseverktøy

Statiske analyseverktøy som benyttes på kildekode, kan håndheve kodestandarder, men om de anvendes på eksisterende kode kan det generere mange meldinger. Advarselsmeldinger stopper ikke koden i å bli oversatt til et utførbart program, men skal ideelt sett bli behandlet slik at vedlikehold av koden blir lettere i framtiden. En gradvis implementasjon med innledende filtre for å ekskludere noen meldinger vil være en effektiv framgangsmåte.

Testadministrasjonsverktøy

Testadministrasjonsverktøy må integreres med andre verktøy eller regneark for å kunne produsere informasjon på et format som organisasjonen trenger.

*6.3 Introduksjon av et verktøy i en organisasjon
(K1)*

15 minutter

Begreper

Ingen spesifikke begreper.

Bakgrunn

Hovedvurderingene i å velge et verktøy for en organisasjon omfatter:

- Vurdering av en organisasjons modenhet, styrker og svakheter og å identifisere muligheter for en forbedret testprosess støttet av verktøy.
- Vurdering opp mot klare krav og objektive kriterier.
- En test (proof-of-concept) ved å bruke testverktøyet under evalueringsfasen for å vurdere om det fungerer effektivt med programvaren som skal testes og med den eksisterende infrastrukturen, eller for å identifisere endringer som behøves i infrastrukturen for å kunne utnytte verktøyet effektivt.
- Evaluering av leverandøren (inkludert opplæring, støtte og kommersielle aspekter) eller av tjenesteleverandørene dersom det ikke er et kommersielt verktøy.
- Identifisering av interne krav til opplæring, støtte og oppfølging i å bruke verktøyet.
- Evaluering av opplæringsbehov i forhold til det nåværende testteamets automatiseringskunnskaper.
- Estimering av kost-nytte forhold basert på konkrete forretningsmål.

Å introdusere det valgte verktøyet i en organisasjon starter med et pilotprosjekt, som har følgende mål:

- Lære flere detaljer om verktøyet.
- Vurdere om verktøyet passer til eksisterende prosesser og praksis, og bestemme hva som må endres.
- Bestemme standard måter å bruke, administrere, lagre og vedlikeholde verktøyet og testmaterialet. (f.eks. bestemme navnekonvensjoner for filer og tester, lage biblioteker og definere modulariteten av testsuiter).
- Vurdere om fordelene vil oppnås til en akseptabel kostnad.

Suksessfaktorer for å ta i bruk et verktøy i en organisasjon er blant annet:

- Trinnvis utrulling av verktøyet til resten av organisasjonen.
- Tilpasse og forbedre prosessene for å passe med bruk av verktøyet.
- Gi undervisning og opplæring/støtte/oppfølging til nye brukere.
- Definere retningslinjer for bruk.
- Implementere en måte å lære av bruken av verktøyet.
- Overvåke verktøybruken og fordeler.
- Gi støtte til testteamet for et gitt verktøy.
- Dele erfaringer med alle team.

Referanser

6.2.2 Buwalda, 2001, Fewster, 1999

6.3 Fewster, 1999

7 Referanser

7.1 Standarder

ISTQB Glossary of terms used in Software Testing Version 2.0 (Norwegian Testing Board norsk- engelsk versjon)

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

Se avsnitt 2.1

[IEEE 829] IEEE Std 829™ (1998/2007) IEEE Standard for Software Test Documentation (ny endret utgave kommer trolig i 2008)

Se avsnitt 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 avsnitt 3.2

[IEEE 12207] IEEE 12207/ISO/IEC 12207-1996, Software Life Cycle Processes

Se avsnitt 2.1

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

Se avsnitt 2.3

7.2 Bøker

[Beizer, 1990] Beizer, B. (1990) *Software Testing Techniques* (2nd edition), Van Nostrand Reinhold: Boston

Se avsnitt 1.2, 1.3, 2.3, 4.2, 4.3, 4.4, 4.6

[Black, 2001] Black, R. (2001) *Managing the Testing Process* (2nd edition), John Wiley & Sons: New York

Se avsnitt 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 avsnitt 6.2

[Copeland, 2004] Copeland, L. (2004) *A Practitioner's Guide to Software Test Design*, Artech House: Norwood, MA

Se avsnitt 2.2, 2.3, 4.2, 4.3, 4.4, 4.6

[Craig, 2002] Craig, Rick D. and Jaskiel, Stefan P. (2002) *Systematic Software Testing*, Artech House: Norwood, MA

Se avsnitt 1.4.5, 2.1.3, 2.4, 4.1, 5.2.5, 5.3, 5.4

[Fewster, 1999] Fewster, M. and Graham, D. (1999) *Software Test Automation*, Addison Wesley: Reading, MA

Se avsnitt 6.2, 6.3

[Gilb, 1993]: Gilb, Tom and Graham, Dorothy (1993) *Software Inspection*, Addison Wesley: Reading, MA

Se avsnitt 3.2.2, 3.2.4

[Hetzel, 1988] Hetzel, W. (1988) *Complete Guide to Software Testing*, QED: Wellesley, MA

Se avsnitt 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. and Pettitcord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York

Se avsnitt 1.1, 4.5, 5.2

[Myers 1979] Myers, Glenford J. (1979) *The Art of Software Testing*, John Wiley & Sons: New York

Se avsnitt 1.2, 1.3, 2.2, 4.3

[van Veenendal, 2004] van Veenendal, E. (ed.) (2004) *The Testing Practitioner* (Chapters 6, 8, 10),
UTN Publishers: The Netherlands

Se avsnitt 3.2, 3.3

8 Vedlegg A – Bakgrunn til pensum

8.1 Historien til dette dokumentet

Dette dokument ble utviklet mellom 2004 og 2007 av en arbeidsgruppe av medlemmer som var utpekt av International Software Testing Qualifications Board (ISTQB). Opprinnelig ble det gransket av en utvalgt granskningsgruppe, og så av representanter fra de som internasjonalt arbeider med test av programvare. Reglene som er brukt under produksjon av dette dokument er vist i vedlegg C.

Dette dokument er pensum for grunnnivået i “International Foundation Certificate in Software Testing”, og første nivå av internasjonal kvalifikasjon som er anerkjent av ISTQB (www.istqb.org).

8.2 Mål med kvalifikasjonen med “Foundation Certificate”

- Å få anerkjennelse for testing som et nødvendig og profesjonelt spesialfelt innen programvareutvikling.
- Å gi et standard rammeverk for utviklingen av en testers karriere.
- Å bidra til at profesjonelt kvalifiserte testere blir anerkjent av arbeidsgivere, kunder og arbeidskolleger, og å øke testernes synlighet.
- Å sørge for utbredelse av sammenlignbar og god testpraksis innen alle områder av programvareutvikling.
- Å identifisere tema innen testing som er relevante og har en verdi i programvaremiljø.
- Å muliggjøre for firma som utvikler programvare å ansette sertifiserte testere og dermed å få kommersielle fordeler over deres konkurrenter ved at de annonserer deres ansettelseskrav for testere.
- Å gi en mulighet for testere og de som har interesse for testing å få en internasjonalt anerkjent kvalifikasjon i fagområdet.

8.3 Mål med den internasjonale kvalifikasjonen (tatt fra ISTQB møte på Sollentuna, november 2001)

- Å kunne sammenligne evner innen testing mellom forskjellige land.
- Å muliggjøre for testere å enklere flytte seg mellom land.
- Å muliggjøre at multinasjonale/internasjonale prosjekter har en felles forståelse av utfordringer innen testing.
- Å øke antallet kvalifiserte testere i verden.
- Å ha større gjennomslagskraft og verdi som et internasjonalt basert initiativ enn som et initiativ basert på et enkelt land.
- Å utvikle en felles internasjonal samling av forståelse og kunnskap om testing gjennom pensum og terminologilisten, og å øke kunnskapsnivået om testing for alle deltakere.
- Å tilby testing som et profesjonelt yrke i flere land.
- Å muliggjøre for testere å få en anerkjent kvalifikasjon på sine nasjonale språk.
- Å muliggjøre deling av kunnskap og ressurser over landegrensene.
- Å sørge for internasjonal anerkjennelse av testere og denne kvalifiseringen gjennom deltakelse fra mange land.

8.4 Startkrav for denne kvalifikasjonen

Startkriteriet for å ta eksamen for ISTQB Foundation Certificate in Software Testing er at kandidaten har en interesse for test av programvare. Men det anbefales på det sterkeste at kandidater også:

- Har i hvert fall en minimal bakgrunn enten i utvikling av programvare eller i testing av programvare, som for eksempel seks måneders erfaring som system- eller akseptansetester eller som utvikler av programvare.

- Tar et kurs som har blitt akkreditert i henhold til ISTQB-standarter (av et av de ISTQB-
anerkjente nasjonale Boards).

8.5 Bakgrunn og historie for grunnsertifikatet (*Foundation Certificate in Software Testing*)

Uavhengig sertifisering av testere av programvare startet i Storbritannia med British Computer Society Information Systems Examination Board (ISEB). Et "Software Testing Board" ble opprettet i 1998 (www.bcs.org.uk/iseb). I 2002 tok ASQF i Tyskland initiativ til en tysk sertifisering av programvaretestere (www.asqf.de). Pensum i dette dokumentet er basert på ISEB og ASQF sine; den inkluderer reorganisert, oppdatert og noe nytt innhold, og vekten er lagt på de tema som gir mest praktisk nytteverdi og hjelp for testere.

Et eksisterende grunnlagssertifikat (Foundation Certificate) fra for eksempel ISEB, ASQF eller et ISTQB-
anerkjent nasjonalt Board, som er gitt før dette internasjonale sertifikat ble utgitt, blir ansett som ekvivalent til det internasjonale sertifikatet. Grunnlagssertifikatet (Foundation Certificate) utløper ikke og krever ikke fornyelse. Datoen et sertifikat ble gitt er vist på sertifikatet.

I hvert deltakende land blir lokale forhold tatt hensyn til av et nasjonalt ISTQB-
anerkjent Software Testing Board. Forpliktelsene disse nasjonale Boards skal oppfylle, er spesifisert av ISTQB, men er implementert av hvert nasjonale Board. Forpliktelsene til et lands Board inneholder akkreditering av kursholdere og arrangering av eksamen.

9 Vedlegg B – Læremål / kunnskapsnivå

Følgende læremål er definert som tilhørende dette pensum. Hvert tema i pensum kan bli prøvd i eksamen tilsvarende sitt læremål.

9.1 Nivå 1: Husk (K1)

Kandidaten gjenkjenner, husker eller kan gjenta en term eller et konsept.

Stikkord: Huske, gjenkjenne, gjenta, observere, vite

Eksempel

Kan gjenkjenne definisjonen “failure” som:

- Avvik eller fravikelse av en komponent eller et system fra dens forventede resultat, tjeneste eller output.
- Hendelse der systemet eller komponenten ikke klarer å utføre sin spesifiserte funksjon eller å utføre den innenfor de begrensninger som er spesifisert.

9.2 Nivå 2: Forstå (K2)

Kandidaten kan velge begrunnelsene eller erklæringene for påstander om dette tema og kan oppsummere, sammenligne, klassifisere, kategorisere og gi eksempler for dette testkonseptet.

Stikkord: oppsummere, generalisere, abstrahere, klassifisere, sammenligne, kartlegge, kontrastere, eksemplifisere, tolke, oversette, representere, trekke slutninger, konkludere, kategorisere, konstruere modeller

Eksempler

Kan forklare grunnen til hvorfor testene bør bli utviklet så tidlig som mulig:

- For å finne feil når de er billigere å ta bort.
- For å finne de viktigste feilene først.

Kan forklare likhetene og forskjellene mellom integrasjon og systemtesting:

- Likheter: tester mer enn en komponent og kan teste ikke-funksjonelle aspekter.
- Forskjeller: integrasjonstesting konsentrerer seg om grensesnitt og interaksjoner, mens systemtesting konsentrerer seg om aspektene som vedrører hele systemet, som prosessering fra start til slutt.

9.3 Nivå 3: Bruke (K3)

Kandidaten kan velge den rette anvendelsen av et konsept eller en teknikk og anvende den for en gitt oppgave.

Stikkord: Implementere, utføre, bruke, følge en prosedyre, tilføre en prosedyre

Eksempel

- Kan identifisere grenseverdier for gyldige og ugyldige partisjoner.
- Kan velge testtilfeller for et gitt tilstandsdiagram for å dekke alle overganger.

9.4 Nivå 4: Analysere (K4)

Kandidaten kan separere informasjon relatert til en prosedyre eller teknikk til dens bestanddeler for bedre forståelse og kan skille mellom fakta og forstyrrende elementer. Dette vil typisk innebære å

analysere et dokument, en programvare eller en prosjektsituasjon og foreslå passende aktiviteter for å løse et problem eller en oppgave.

Stikkord: Analysere, organisere, finne sammenheng, integrere, angi hovedpunktene, analysere syntaktisk, strukturere, finne kjennetegn, dekonstruere, differensiere, se forskjell på, finne forskjeller, fokusere, velge

Eksempel

- Analysere produktrisiko og foreslå forebyggende og korrigerende migrasjonsaktiviteter
- Beskrive hvilke deler av en hendelsesrapport som er fakta og hvilke som avviker fra resultater

Referanser

(For de kognitive nivåer av læremål)

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

10 Vedlegg C – Regler som er brukt av ISTQB

Grunnivå-pensum (Foundation syllabus)

Reglene listet opp her ble brukt under utvikling og granskning av dette pensum. En "TAG" er vist etter hver regel som en kort kode av regelen.)

Generelle regler

- SG1. Pensum bør være forståelig og lærbar for personer med 0 til 6 måneder (eller mer) erfaring i testing. (6-MONTH)
- SG2. Pensum bør være praktisk heller enn teoretisk. (PRACTICAL)
- SG3. Pensum bør være klart og entydig for dens planlagte lesere. (CLEAR)
- SG4. Pensum bør være forståelig for personer fra ulike land, og lett å oversette til ulike språk. (TRANSLATABLE)
- SG5. Det originale pensum (syllabus) skal bruke Amerikansk Engelsk. (AMERICAN-ENGLISH)

Aktuelt innhold

- SC1. Pensum bør inneholde moderne testkonsepter og bør reflektere aktuell beste praksis i test av programvare der dette er generelt akseptert. Pensum skal oppdateres hvert tredje til femte år. (RECENT)
- SC2. Pensum bør minimere nåtidsrelaterte ting, som dagens markedssituasjon, for å muliggjøre at det fortsatt er aktuelt etter tre til fem år (SHELF-LIFE).

Læremål

- LO1. Læremål bør skille mellom ting som skal gjenkjennes/huskes (kognitivt nivå K1), ting kandidaten skal forstå konseptet av (K2), ting som kandidaten skal kunne anvende (K3) og ting som kandidaten skal kunne bruke for å analysere et dokument, programvare eller prosjektsituasjon i sin sammenheng (K4). (KNOWLEDGE-LEVEL)
- LO2. Beskrivelsen av innholdet bør være sammenhengende med læremålene. (LO-CONSISTENT)
- LO3. For å illustrere læremålene, bør eksempler på eksamensspørsmål publiseres sammen med pensum. (LO-EXAM)

Totalstruktur

- ST1. Strukturen til pensum bør være klar og tillate kryssreferering til og fra andre deler, fra eksamensspørsmål og fra andre relevante dokumenter. (CROSS-REF)
- ST2. Overlapp mellom avsnitt i pensum avsnitt bør være minimale. (OVERLAP)
- ST3. Hvert avsnitt i pensum bør ha samme struktur. (STRUCTURE-CONSISTENT)
- ST4. Pensum bør inneholde versjon, publiseringsdato og sidenummer på hver side. (VERSION)
- ST5. Pensum bør inkludere retningslinjer for hvor mye tid som skal brukes for hvert avsnitt i løpet av et kurs (for å vise den relative viktigheten av hvert tema). (TIME-SPENT)

Referanser

- SR1. Kilder og referanser blir gitt for hvert konsept i pensum for å hjelpe kursholdere til å finne mer informasjon om temaet. (REFS)
- SR2. Der det ikke er direkte identifiserte og klare kilder, skal pensum inneholde mer detaljer. For eksempel: definisjoner er i terminologilisten, derfor blir bare begrepene listet i pensum. (NON-REF DETAIL)

Informasjonskilder

Begreper brukt i dette pensum er definert i ISTQB Glossary Of Terms Used In Software Testing. En versjon av denne er tilgjengelig på engelsk på www.istqb.org (websidene til ISTQB). En norsk-engelsk versjon er tilgjengelig på www.istqb.no .

En liste over anbefalte bøker om test av programvare blir også distribuert parallelt til dette pensum. Disse bøkene ligger i referanselisten i dette dokumentet.

11 Vedlegg D – Informasjon til kursholdere

Hvert hovedkapittel i pensum er tildelt et tidsrom i minutter. Målet med dette er både å gi veiledning angående den relative andelen tid som skal brukes for hver del av et akkreditert kurs, og å gi en omtrentlig minimaltid for læringen av hvert kapittel. Kursholdere kan bruke mer tid enn vist, og kandidater kan igjen bruke mer tid for å lese og utforske området. Et kurs behøver heller ikke følge samme rekkefølgen som dette pensum.

Pensum inneholder referanser til etablerte standarder som skal brukes under forberedelsen av opplæringsmateriellet. Hver standard må siteres med angivelse av versjon. Andre publikasjoner, maler eller standarder som ikke er referert i dette pensum, kan også brukes og refereres, men de vil ikke bli grunnlag for eksamen.

Følgende områder av pensum krever praktiske øvelser:

4.3 Spesifikasjonsbaserte eller black-box teknikker

Praktisk arbeid (korte øvelser) bør inkluderes for å dekke følgende fire teknikker: ekvivalensklasseinndeling, grenseverdianalyse, beslutningstabelltesting og tilstandsovergangstesting. Leksjonene og øvelsene som tilhører disse teknikkene bør baseres på det som er brukt for å forklare hver teknikk.

4.4 Strukturbaserte eller white-box teknikker

Praktisk arbeid (korte øvelser) bør inkluderes for å sjekke om en test klarer å oppnå 100% programinstruksjons- og 100% beslutningsdekning, og for å konstruere testtilfelle for å dekke gitte kontrollflyt.

5.6 Problemhåndtering

Praktisk arbeid (korte øvelser) bør inkluderes for å dekke skrivning og/eller kontroll av en problemrapport.

12 Appendix E – Release Notes Syllabus 2010

1. Changes to Learning Objectives (LO) include some clarification
 - a. Wording changed for the following LOs (content and level of LO remains unchanged): LO-1.2.2, LO-1.4.1, LO-2.1.1, LO-2.1.3, LO-4.6.1, LO-6.3.2
 - b. K4 has been added. Reason: some requirements (LO-4.4.4 and LO-5.6.2) have already been written in a K4 manner, and LO-4.6.1 questions are easier to write and examine on K4 level.
 - c. LO-1.1.5 has been reworded and upgraded to K2. Because a comparison of terms of defect related terms can be expected.
 - d. LO-1.2.3 Explain the difference between the two activities debugging and testing is a new LO. The content was already covered.
 - e. LO-3.1.3 Comparison issues covered
 - f. LO-3.1.4 removed. Partly redundant with LO-3.1.3.
 - g. LO-3.2.1 Consistency with content.
 - h. LO-3.3.2 upgraded to K2 in order to be consistent with LO-3.1.2
 - i. LO-6.1.2 removed as it is part of LO-6.1.3, which has been reworded due to nonappropriate use of a K2 keyword
2. Consistent use for test approach according to the definition in the glossary. The term test strategy will not be required as term to recall.
3. Chapter 1.4 now contains the concept of traceability between test basis and test cases.
4. Chapter 2.x now contains test objects and test basis.
5. Re-testing is now the main term as in the glossary and not confirmation testing.
6. The aspect data quality and testing has been added at several locations in the syllabus: data quality and risk in Chapter 2.2, 5.5, 6.1.8
7. Chapter 5.2.3 Entry Criteria are added as a new subchapter. Reason: Consistency to Exit Criteria (-> entry criteria added to LO-5.2.9).
8. Consistent use of the terms test strategy and test approach with their definition in the glossary.
9. Chapter 6.1 shortened because the tool descriptions were too large for a 45 minute lesson.
10. IEEE Std 829:2008 has been released. This version of the syllabus does not yet consider this new edition. Section 5.2 refers to the document Master Test Plan. The content of the Master Test Plan is covered by the concept that the document "Test Plan" covers different levels of planning: Test plans for the test levels can be created as well as a test plan on the project level covering multiple test levels. Latter is named Master Test Plan in this syllabus and in the ISTQB Glossary.
11. Code of Ethics has been moved from the CTAL to CTFL.