Certifierad testare
Grundnivå
Kursplan

Version 2011

Swedish Software Testing Board
International Software Testing Qualifications Board

V2.2

*Senaste versionen av detta dokument kan hämtas från **http://www.sstb.se**. Versioner av detta dokument hämtade från andra webbsidor än **http://www.sstb.se** kan vara en äldre version.*

Detta dokument baseras på ISTQBs dokument “Certified Tester Foundation Level Syllabus” och har översatts till svenska av Swedish Software Testing Board, SSTB (Ingvar Nordström (ordförande), Beata Karpinska, Klaus Zeuge, Ingalill Hall, Robert Bornelind, Maria Jönsson, Mats Grindal, Johan Klintin, Patrik Norrby, Anders Pettersson). Dokumentet motsvarar den internationella certifieringen “ISTQB® Cerified Tester Foundation Level” vars copyright tillsvidare ägs med ensamrätt av författarna (Thomas Müller (ordförande), Armin Beer, Martin Klonk, Rahul Verma). Författarna överlåter härmed upphovsrätten till International Software Testing Qualifications Board (dvs. den internationella delen av ISTQB). Författarna och ISTQB har kommit överens att följande villkor skall gälla:

1) Individer och kursarrangörer får använda denna kursplan som bas för kurser om referenser ges till författarna, ISTQB och SSTB som copyright-ägare av dokumentet och att all annonsering av kurser först kan omnämna kursplanen efter det att en officiell ackreditering av kursmaterialet har gjorts av SSTB (ett av ISTQB erkänt nationellt organ).
2) Varje individ eller grupp av individer får använda denna kursplan som bas för artiklar, böcker eller andra skrifter om författarna, ISTQB och SSTB är nämnda som källan och copyright-ägare av kursplanen.
3) Varje av ISTQB erkänd nationell styrelse, har rättighet att översätta och licensiera kursplanen och dess översättning till andra kurshållare.

SSTB, Swedish Software Testing Board, är av ISTQB en erkänd nationell styrelse och har därmed övertagit rättigheterna för den svenska översättningen.

Versionshistorik

Version	Datum	Kommentarer
SSTB 2005-2006 (1.1)	2006-03-07	- Anmärkningar från Anders P, - efter första ackreditering, anmärkningar genomgångna) - rättning av stavfel,
SSTB 2007 (2.0)	2007-12-20	Versionsnummer ändrat till 2.0 (internt) Dokumentnamn ändrat till version 2007 (överenstämmer med engelska versionen) Åtgärdat inkomna kommentarer. Harmonisering med ”Syllabus CTFL ver. 2007”. Ändring av indexposter.
SSTB 2010 (2.1)	2010-10-26	Första version av översättning av ändringar introducerad i Syllabus 2010 inkl. ”Errata to CTFL 2010 8-Aug-10”
SSTB 2011 (2.2)	2011-04-29	Uppdaterad efter ISTQB CTFL Syllabus 2011

Innehåll

Tillkännagivanden

Det engelska originalet till detta dokument har producerats av ISTQBs arbetsgrupp för grundnivån (version 2011): Thomas Müller (ordförande), Debra Friedenberg. Arbetsgruppen vill speciellt tacka granskarna Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal samt alla nationella styrelser för deras förslag.

Det engelska originalet till detta dokument har producerats av ISTQBs arbetsgrupp för grundnivån (version 2010): Thomas Müller (ordförande), Rahul Verma, Martin Klonk and Armin Beer. Arbetsgruppen vill speciellt tacka granskarna Rex Black, Mette Bruhn–Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal samt alla nationella styrelser för deras förslag.

ISTQBs arbetsgrupp för grundnivån (version 2007): Thomas Müller (ordförande), Dorothy Graham, Debra Friedenberg, and Erik van Veenendaal och granskningsgruppen (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, and Wonil Kwon) samt alla nationella styrelser för deras förslag.

ISTQBs arbetsgrupp för grundnivån (version 2005): Thomas Müller (chair), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson and Erik van Veenendaal samt alla nationella styrelser för deras förslag.

Bidrag till den svenska översättningen:
För den svenska översättningen har SSTB stått, speciellt Sigrid Eldh, Ingvar Nordström, Maria Jönsson, Ingela Skytte, Kennet Osbjer, Klaus Zeuge, Johan Klintin, Anders Pettersson och Mats Grindal.
Dessutom vill vi tacka Anders Pettersson, Daniel Sundmark, Sigrid Eldh, samt Mattias Nordin, Ingvar Nordström, Ingela Skytte, Beata Karpinska Säther, Klaus Zeuge och Mats Grindal för terminologiöversättningen som fungerat som bas för denna översättning.

Introduktion till denna kursplan

Avsikten med detta dokument

Det engelska originalet till denna kursplan utgör grunden för den Internationella programtestcertifieringen på grundnivå. ISTQB (International Software Testing Qualifications Board) erbjuder denna kursplan till nationella styrelser, att ackreditera kurshållare och erbjuda examinationsfrågor på det nationella språket. SSTB (Swedish Software Testing Board), är den svenska motsvarigheten och är ansvarig att ackreditera och erbjuda examinationsfrågor och examinationstillfällen på svenska. Kursleverantörer avgör undervisningsmetod och producerar utbildningsmaterial för ackreditering. Kursplanen hjälper kandidaterna för att underlätta förberedelserna till examineringen.

Historia och bakgrund till denna kursplan kan hittas i Bilaga A.

Certifierad testare på grundnivå i programvarutestning

Målgruppen för certifiering på grundnivån är den som är på något sätt involverad i programvaruutveckling och programvarutestning. Den syftar i första hand till människor i roller som testare, testanalytiker, testingenjörer, testkonsulter, testledare, acceptanstestare och programvaruutvecklare. Certifiering på grundnivå är också användbar för de som vill ha en grundläggande baskunskap om programvarutestning, t.ex. projektledare, kvalitetsansvariga, systemansvariga, utvecklingsansvariga, IT chefer, produkt och marknadsanalytiker och chefskonsulter. Att vara certifierad testare på grundnivå i programvarutestning innebär också att man kan fortsätta till högre certifieringsnivåer inom programvarutestområdet.

Inlärningsmål/kognitiv kunskapsnivå

De kognitiva nivåerna är angivna i varje kapitel i kursplanen och klassificeras enligt följande:

- K1: komma ihåg
- K2: förstå
- K3: använda
- K4: analysera

Mer detaljer och exempel på inlärningsmål finns i bilaga B.

Alla begrepp listade under “Begrepp” precis under varje kapitelrubrik skall kännas igen (K1) även om de inte tydligt uttrycks i inlärningsmålen.

Examinering

Certifikatet på grundnivå baseras på denna kursplan. Svaren på frågorna kan innebära att information behöver hämtas från mer än ett kapitel i kursplanen.

Alla kapitel i kursplanen är föremål för examinering. De frågor som används i examinationen är av typen flerval. Examen kan tas som en del av en ackrediterad kurs eller oberoende. t.ex. på något examinationscenter eller en publik examination. Det är inget krav på att ha gått en ackrediterad kurs för att få deltaga i en examinering.

Ackreditering

SSTB, som är erkänt av ISTQB, är den instans som ackrediterar kursleverantörer, vars kursmaterial följer denna kursplan. Ackrediteringsföreskrifter och hänvisningar kan erhållas av SSTB eller från ISTQB. En ackrediterad kurs innebär att kursleverantören erkänner och följer SSTBs föreskrifter och hänvisningar samt att de följer kursplanen. Detta innebär också att de kan hålla en ISTQB-examination som del av kursen. Mer information finns i bilaga D.

Detaljnivå

Detaljnivån i denna kursplan tillåter internationell jämförbar undervisning och examination. För att uppnå detta mål, omfattar kursplanen följande:

- Generella undervisningsmål som beskriver avsikten med grundnivån
- En lista med information som skall läras ut och som innehåller beskrivningar och referenser till extra material och källor.
- Inlärningsmål för varje kunskapsområde som beskriver de kognitiva kunskapsmålen och vad man avser med denna kunskap.
- En lista av begrepp som studenterna måste kunna komma ihåg och förstå.
- En beskrivning av nyckelområden att lära ut, vilket också inbegriper erkända källor och standarder.

Kursplanens innehåll är inte en beskrivning av hela kunskapsområdet programvarutestning, utan speglar den kunskapsnivå som måste täckas in av kursen för grundnivån.

Hur kursplanen är organiserad

Kursplanen innehåller 6 kapitel. Översta rubriken för varje område visar vilka inlärningsmål som täcks av kapitlet och specificerar också tiden för varje kapitel. T.ex.:

2. Testning genom programvarans livscykel (K2)	115 minuter

Rubriken visar att kapitel 2 har inlärningsmål för K1 och K2 (men inte K3). K1 förutsätts gälla därför att kapitlet riktar sig mot en högre nivå. Avsikten är att det tar ungefär 115 minuter att undervisa materialet i kapitlet. Varje kapitel innehåller ett antal avsnitt som också har inlärningsmål och ungefärlig tidsomfattning beskriven. De underavsnitt som inte har tidsangivelser specifikt utskrivna inkluderas i tiden för överordnat avsnitt.

Historien bakom detta dokument

Originaldokumentet på engelska som är basen för detta dokument togs fram under 2004-2005 av en arbetsgrupp som utpekats av ISTQB. Dokumentet har sedan genomgått en granskningsprocess med representanter från den internationella testvärlden. Regler som används för att producera detta dokument beskrivs i bilaga C.

Ovanstående kursplan har därefter översatts av SSTB under perioden augusti till oktober 2005. Den har därefter genomgått granskning av flera representanter från olika företag och organisationer i Sverige.

Detta dokument är kursplanen för den första nivån, grundnivån av den internationellt erkända certifieringen, godkänd av ISTQB (**www.istqb.org**) via sin svenska organisation SSTB (**www.sstb.se**). De länder som var medlemmar i ISTQB när detta dokument framtogs var Österrike, Danmark, Finland, Frankrike, Tyskland, Indien, Israel, Japan, Korea, Norge, Polen, Portugal, Spanien, Sverige och Schweiz, Nederländerna, England och USA.

1. Grunderna inom test (K2) — 155 minuter

Inlärningsmål för grunderna inom test

Följande mål visar vad som uppnås efter avslutande av respektive modul.

1.1. Varför är det nödvändigt att testa? (K2)

IM-1.1.1. Beskriva, med exempel, på vilket sätt ett fel i ett program kan orsaka skada för en person, för miljön eller för ett företag. (K2)
IM-1.1.2. Skilja mellan grundorsaken till ett fel och dess effekter. (K2)
IM-1.1.3. Ange anledningar till varför test är nödvändigt genom att ge exempel. (K2)
IM-1.1.4. Beskriva varför test är en del av kvalitetssäkringen och ge exempel på hur test bidrar till högre kvalitet. (K2)
IM-1.1.5. Förklara och jämföra begreppen misstag, defekt, fel, felsymptom och de motsvarande begreppen mänskligt misstag och bugg med hjälp av exempel. (K2)

1.2. Vad är testning? (K2)

IM-1.2.1. Komma ihåg de vanliga målen med testning. (K1)
IM-1.2.2. Ge exempel på syften med testning i olika faser i programvarans livscykel. (K2)
IM-1.2.3. Skilja på testning och avlusning. (K2)

1.3. Sju testprinciper (K2)

IM-1.3.1. Förklara de sju grundläggande principerna inom test. (K2)

1.4. Grundläggande testprocess (K1)

IM-1.4.1. Komma ihåg de fem grundläggande testaktiviteterna från planering till testavslutsaktiviteter och huvuduppgifterna för varje aktivitet. (K1)

1.5. Testningens psykologi (K2)

IM-1.5.1. Komma ihåg de psykologiska faktorer som påverkar framgången för testningen (K1):
IM-1.5.2. Jämföra tankesätten hos en testare och en utvecklare. (K2)

1.1 *Varför är test nödvändigt (K2)*	*20 minuter*

Begrepp
Bugg, defekt, misstag, fel, felsymptom, mänskligt misstag, kvalitet, risk

1.1.1 Programvarusystemen i ett större sammanhang (K1)

Programvarusystem är en integrerad del av livet, från verksamhetstillämpningar (t.ex. bankverksamhet) till konsumentprodukter (t.ex. bilar). De flesta har upplevt att programvaran inte fungerar som man förväntat sig. Programvara som inte fungerar korrekt kan leda till många problem, t.ex. förlust av pengar, tid eller affärsmässigt rykte och kan till och med orsaka skada eller dödsfall.

1.1.2 Orsaker till programvarufel (K2)

En människa kan göra ett misstag (mänskligt fel), som leder till ett fel (defekt, bugg) i programvaran eller i ett dokument. Om ett fel i koden exekveras, kommer kanske systemet inte att lyckas göra vad det skall (eller göra något det inte ska) och orsaka ett felsymptom. Fel i program, system eller dokument kan orsaka felsymptom men alla fel behöver inte göra det.

Fel uppstår därför att människor gör misstag på grund av tidspress, komplex kod, komplexitet i infrastrukturen, förändrad teknologi och/eller att många system interagerar.

Felsymptom kan även orsakas av miljömässiga förhållanden t.ex. strålning, magnetism, elektriska fält eller föroreningar. De kan orsaka fel i inbyggda program eller påverka exekveringen av program genom att förändra hårdvarans förutsättningar.

1.1.3 Testningens roll inom programvaruutveckling, underhåll och drift (K2)

Rigorös testning av system och dokumentation kan hjälpa till att reducera risken för att problem uppstår under drift men också bidra till programvarusystemets kvalitet om fel rättas innan systemet frisläpps för användning i drift.

Det kan också krävas test av programvara för att uppnå kontraktsmässiga eller legala krav eller för att uppfylla industrispecifika standarder.

1.1.4 Test och kvalitet (K2)

Med hjälp av test är det möjligt att mäta kvaliteten på programvaran i termer av hittade fel för både funktionella och icke-funktionella programvarukrav och egenskaper (t.ex. pålitlighet, användbarhet, effektivitet, underhållbarhet och portabilitet). För mer information om icke-funktionella tester se kapitel 2; för mer information om egenskaper hos programvara se standarden ’Software Engineering – Software Product Quality’ (ISO 9126).

Test kan ge ett förtroende för programvarans kvalitet om inga eller få fel upptäcks. Ett korrekt designat testfall som godkänns minskar den totala risknivån för ett system. När test hittar fel och dessa åtgärdas, ökar kvaliteten hos programvarusystemet.

Erfarenheter från tidigare projekt bör tas tillvara. Genom att förstå de grundorsakerna till de hittade felen i andra projekt kan processerna förbättras vilket i sin tur bör förhindra att dessa fel dyker upp igen och, som en konsekvens, förbättra kvaliteten hos framtida system. Detta är en aspekt i arbetet att kvalitetssäkra programvaran.

Test bör vara en del av de kvalitetssäkrande aktiviteterna, tillsammans med utvecklingsstandarder, utbildning och felanalys.

1.1.5 Hur mycket testning behövs? (K2)

Beslut hur mycket testning som behövs bör ta hänsyn till risknivån, inklusive tekniska, säkerhetsrelaterade och affärsmässiga risker och projektbegränsningar som tid och budget. (För ingående diskussioner om risker, se kapitel 5).

En utförd testning bör ge tillräcklig information för beslutsunderlag. Detta för att intressenter kan fatta beslut om frisläppande av programvaran eller systemet, beslut om att gå vidare till nästa utvecklingssteg eller beslut om överlämnande till kund.

1.2 *Vad är testning (K2)*	*20 minuter*

Begrepp
Avlusning, krav, granskning, testfall, testning, testmål.

Bakgrund
En vanlig uppfattning om testning är att det bara består av att exekvera testfall, dvs. exekvera programvaran. Det är visserligen en del av testningen, men fler testaktiviteter ingår också.

Testning innefattar en lång rad aktiviteter, såsom planering och styrning, val av testvillkor, design och exekvering av testfall och kontroll av resultat, utvärdering av avslutskriterier, rapportering om testprocessen och systemet som testas och avslutning eller stängning (t.ex. efter det att en testfas avslutats). Test inkluderar också granskning av dokument (även källkod) och genomförande av statisk analys.

Både dynamisk och statisk testning kan användas som medel för att uppnå likadana mål och de tillhandahåller information som kan användas för att förbättra systemet som testas men även utvecklings- och testprocesserna.

Det kan finnas olika mål för test:
- att hitta fel
- att få förtroende för kvalitetsnivån
- att tillhandahålla information för att kunna ta beslut
- att förebygga fel

Tankeprocessen och aktiviteterna som ingår i att designa tester tidigt i livscykeln (verifiering av testbasen genom testdesign) kan hjälpa till att förhindra att fel introduceras i koden. Granskning av dokument (t.ex. kravspecifikationer) och identifiering och lösning av avvikelser hjälper också till att förhindra att fel uppstår i koden.

Olika synvinklar inom test kan ge upphov till olika mål. Det huvudsakliga målet vid test i samband med utveckling (t.ex. komponent-, integrations- och systemtest) kan vara att provocera så många felsymptom som möjligt så att fel kan identifieras och åtgärdas. I acceptanstest, kan det huvudsakliga målet vara att bekräfta att systemet fungerar som förväntat, för att få förtroende om det uppfyller ställda krav. I vissa fall kan det huvudsakliga målet för test vara att utvärdera kvaliteten hos programvaran (utan avsikt att åtgärda felen), att ge information till intressenter om risken med att frisläppa systemet vid ett givet tillfälle. Underhållstest inkluderar ofta testning för att konfirmera att inga nya fel har introducerats när ändringarna infördes. Vid driftstestning, kan det huvudsakliga målet vara att utvärdera systemets egenskaper såsom tillförlitlighet eller tillgänglighet i en driftsliknande miljö.

Avlusning och test är olika saker. Dynamisk testning kan visa på felsymptom som orsakas av fel. Avlusning är den utvecklingsaktivitet som identifierar, analyserar och avlägsnar orsaken till felsymptomet. Efterföljande omtest av rättningar av en testare försäkrar att rättningen verkligen åtgärdade felsymptomen. Ansvaret för varje aktivitet är vanligtvis att testare testar och utvecklare avlusar.

Testprocessen och dess aktiviteter förklaras i kapitel 1.4.

1.3 *Sju testprinciper (K2)*	*35 minuter*

Begrepp
Uttömmande testning

Principer
Ett antal testprinciper har föreslagits under de senaste 40 åren och de erbjuder allmänna riktlinjer som är gemensamma för all testning.

Princip 1 – Test visar att det finns fel
Test kan visa att fel finns men kan inte visa att det inte finns några fel. Test minskar sannolikheten för att oupptäckta fel finns i programvaran men, även om inga fel hittas, så är det inget bevis för att programvaran är felfri.

Princip 2 – Uttömmande testning är omöjlig
Att testa allt (alla kombinationer av inmatningar och förutsättningar) är inte lämpligt, förutom i triviala fall. Istället för uttömmande testning används risker och prioriteringar för att fokusera testansträngningarna.

Princip 3 – Tidig testning
För att hitta fel tidigt skall testaktiviteterna påbörjas så tidigt som möjligt i programvarans eller systemets livscykel och fokuseras på definierade mål.

Princip 4 – Ansamlingar av fel
Testinsatsen skall fokuseras proportionellt mot den förväntade, och senare konstaterade, feltätheten hos modulerna. Ett litet antal moduler innehåller de flesta fel som upptäcks vid testning före frisläppande eller är upphov till flest felsymptom i drift.

Princip 5 – Immunitets- paradoxen
Om samma tester upprepas gång på gång så kommer samma uppsättning av testfall till slut inte att upptäcka några nya fel. För att övervinna denna ”immunitets-paradox” behöver testfallen regelbundet granskas och revideras och annorlunda tester behöver skrivas för att motionera olika delar av programvaran eller systemet för att möjligen hitta fler fel.

Princip 6 – Test beror på sammanhang
Testning sker på olika sätt i olika sammanhang. Säkerhetskritisk programvara testas till exempel inte på samma sätt som ett e-handels system.

Princip 7 – Frånvaro-av-fel-fallgropen
Att hitta och åtgärda fel hjälper inte om systemet som byggts är oanvändbart och inte uppfyller användarnas behov och förväntningar.

1.4 *Grundläggande testprocess (K1)*	*35 minuter*

Begrepp
Omtestning av felrättningar, omtestning, avslutskriterier, avvikelse, regressionstestning, testbas, testvillkor, testtäckning, testdata, testexekvering, testlogg, testplan, testprocedur, testpolicy, testsvit, slutlig testrapport, testvara.

Bakgrund
Den mest synliga delen av testprocessen är testexekveringen. Men för att en testprocess ska vara effektiv och ändamålsenlig, bör testplaner även innehålla information om hur mycket tid som skall användas för att planera testerna, designa testfallen, förbereda exekveringen och utvärdera resultaten.

Den grundläggande testprocessen består av följande huvudaktiviteter:
- testplanering och styrning
- testanalys och design
- realiserande och exekvering av tester
- utvärdering av avslutskriterium och rapportering
- testavslutsaktiviteter.

Även om aktiviteterna i processen logiskt sett är sekventiella så kan de överlappa varandra eller ske parallellt. Det är oftast nödvändigt att anpassa dessa huvudaktiviteter till systemets och projektets sammanhang.

1.4.1 Testplanering och styrning (K1)
Testplanering är den aktivitet som görs för att definiera testmålen och specificera övriga testaktiviteterna med syfte att nå målen och att slutföra uppdraget.

Teststyrning är den pågående aktivitet som jämför faktiskt framåtskridande mot plan och rapporterar status och avvikelser från planen. I detta ingår att vidta nödvändiga åtgärder för att slutföra uppdraget och att nå målet för projektet. För att kunna styra testarbetet bör testaktiviteterna övervakas under hela projektet. Testplaneringen tar också hänsyn till återkoppling från övervakning och uppföljande aktiviteter.

Testplanering och teststyrning är beskrivna i kapitel 5 i denna kursplan.

1.4.2 Testanalys och design (K1)

Testanalys och design är den aktivitet där generella testmål omvandlas till påtagliga testvillkor och testfall.

Analys och design har följande huvuduppgifter:

- Att granska testbasen (såsom krav, risknivå[1], riskanalysrapporter, specifikationer, arkitektur, design, gränssnitt).
- Evaluera testbarheten av testbasen och testobjekten
- Att identifiera och prioritera de testvillkor som baserats på analys av testobjekten, specifikationen samt programvarans beteende och struktur.
- Att utveckla och prioritera högnivåtestfall.
- Att identifiera nödvändiga testdata för testvillkor och testfall.
- Att designa hur testmiljön skall sättas upp och att identifiera den infrastruktur och de verktyg som behövs.
- Att skapa en dubbelriktad spårbarhet mellan testbas och testfall.

1.4.3 Realiserande och exekvering av tester (K1)

Realiserande och exekvering av tester är den aktivitet där testprocedur eller skript definieras genom att kombinera testfall i en speciell ordning och inkludera nödvändig information som behövs för exekvering, miljön iordningställs och testerna körs.

Realiserande och exekvering av tester har följande huvuduppgifter:

- Färdigställa, realisera och prioritera testfall (inklusive identifiering av testdata).
- Att utveckla och prioritera testprocedurer, skapa testdata och, valfritt, förbereda testexekveringsplattform samt skriva automatiserade testskript.
- Att skapa testsviter baserade på testprocedur för en effektiv testexekvering.
- Att verifiera att testmiljön är rätt uppsatt.
- Verifiera och uppdatera spårbarheten mellan testbas och testfall.
- Att exekvera testprocedur, antingen manuellt eller genom att använda testexekveringsverktyg, enligt det planerade schemat.
- Att logga utfallet av testexekveringen och spela in identiteter och versioner av programvaran som testas, testverktyg och testvara.
- Att jämföra aktuellt resultat med förväntat resultat.
- Att rapportera skillnader i form av avvikelser och analysera dem med syftet att fastställa orsaken till dem (t.ex. ett fel i koden, i specificerat testdata, i testdokumentet eller ett misstag i utförandet av testen).
- Att repetera testaktiviteterna som ett resultat av gjorda åtgärder för varje avvikelse. Det kan till exempel vara omexekvering av en test som tidigare gått fel med syfte att bekräfta att felet är åtgärdat (omtestning av felrättningar), exekvering av ett korrekt test och/eller exekvering av tester med syfte att försäkra sig om att inga fel har introducerats i oförändrade delar av programvaran eller att felrättningarna inte blottställde andra fel (regressionstestning).

[1] I vilken utsträckning programvaran uppfyller, eller måste uppfylla utvalda programvarubaserade systemegenskaper (t.ex programvarukompexitet, risk, säkerhetsnivå, informationssäkerhetsnivå, önskad prestanda, tillförlitlighet eller kostnad) som har fastställts för att återspegla programvarans betydelse för dess intressenter.

1.4.4 Utvärdering av avslutskriterier och rapportering (K1)

Att utvärdera avslutskriterier är den aktivitet där testexekveringen utvärderas mot de definierade målen. Detta bör göras för varje testnivå.

Utvärdering av avslutskriterier har följande huvuduppgifter:

- Att kontrollera testloggar mot de avslutskriterier som specificerats vid testplaneringen.
- Att bedöma om fler tester behövs eller om avslutskriterierna behöver ändras.
- Att skriva en slutlig testrapport avsedd för intressenterna.

1.4.5 Testavslutsaktiviteter(K1)

Vid testavslut samlas data in från de avslutade aktiviteterna för att sammanfatta erfarenheter, fakta och mätetal och för att arkivera testmaterial. Detta kan ske t.ex. vid frisläppande av ett programvarusystem, vid ett fullbordat (eller avbrutet) testprojekt, när en milstolpe har nåtts eller en underhållsutgåva har blivit färdig.

Vid testavslut ingår följande huvuduppgifter:

- Att kontrollera vilka planerade leverabler som blivit levererade.
- Att kontrollera att avvikelserapporter är stängda och att initiera ändringsbegäran för de som fortfarande är öppna.
- Att kontrollera den dokumentation som visar godkännande av systemet.
- Att göra klart och arkivera testvaran, testmiljön och testinfrastrukturen för senare återanvändning.
- Att lämna över testvaran till förvaltningsorganisationen.
- Att analysera lärdomar för att identifiera nödvändiga förändringar av framtida utgåvor och projekt.
- Att använda insamlad information till att förbättra testmognaden.

1.5 *Testningens psykologi (K2)*	*25 minuter*

Begrepp
Felgissning, oberoende.

Bakgrund
Det tänkesätt som man måste ha när man testar och granskar skiljer sig från den man har när man utvecklar programvara. Med rätt tänkesätt kan utvecklare testa sin egen kod men att lägga detta ansvar på en testare ger ett mervärde. Man får mer fokus och dessutom ytterligare fördelar såsom ett oberoende synsätt som utbildade och professionella testresurser kan erbjuda.
Oberoende testning kan genomföras på vilken testnivå som helst.

En viss grad av oberoende (för att undvika författarens förutfattade mening) gör ofta att testaren blir mer effektiv i att hitta fel och felsymptom. Oberoendet är emellertid inte en ersättning för förtrolighet, utvecklare kan själv hitta många fel i sin egen kod på ett effektivt sätt. Flera nivåer av oberoende kan definieras, listade nedan från lågt till högt:

- Tester utvecklade av personer som skrev den programvara som skall testas (låg nivå av oberoende).
- Tester utvecklade av andra personer (t ex från utvecklingsgruppen).
- Tester utvecklade av personer som tillhör en annan organisatorisk grupp (t.ex. en oberoende testgrupp) eller testspecialister (t.ex. en användbarhetstest- eller prestandatestspecialist).
- Tester utvecklade av personer från en annan organisation eller annat företag (d.v.s. outsourcing eller certifiering av en extern organisation).

Individer och projekt drivs av mål. Individer tenderar till att rätta sina planer efter de mål som ledningen eller andra intressenter satt upp, t.ex. att hitta fel eller att bekräfta att programvaran uppfyller målsättningarna. Det är därför viktigt att tydligt deklarera målen med testningen.

Att identifiera felsymptom under testningen kan uppfattas som kritik mot produkten och mot utvecklaren eller författaren. Testning blir därför ofta sedd som en destruktiv aktivitet trots att det är mycket konstruktivt när det gäller hantering av produktrisker. Att leta efter felsymptom i ett system kräver nyfikenhet, professionell pessimism, ett kritiskt öga, uppmärksamhet på detaljer, god kommunikation med utvecklare och en erfarenhet på vilken man kan basera felgissning.

Om kommunikationen beträffande misstag, defekter, eller felsymptom sker på ett konstruktivt sätt kan dålig stämning mellan testare och analytiker, designers och utvecklare undvikas. Detta gäller defekter som har hittats både genom granskning och genom dynamisk testning.
Testaren och testledaren behöver ha en bra social kompetens för att kommunicera faktamässig information om fel, defekter, avvikelser, framåtskridande och risker på ett konstruktivt sätt. Information om funna fel kan hjälpa författaren av ett dokument till att förbättra sina färdigheter. Fel som hittas och åtgärdas medan man testar kommer senare att spara tid och pengar och även minska riskerna.

Kommunikationsproblem kan uppstå, speciellt om testarna bara ses som budbärare av oönskade nyheter om fel, defekter och avvikelser. Det finns flera sätt att förbättra kommunikationen och förhållandena mellan testare och andra:

- Börja med samarbete istället för strider – påminn alla om det gemensamma målet att få ett system med bättre kvalitet.
- Kommunicera avvikelser hos produkten på ett neutralt, faktafokuserat sätt utan att kritisera personen som skapat den, skriv till exempel objektiva och faktamässiga avvikelserapporter och granska det som hittas.
- Försök att förstå hur den andra personen känner sig och varför han/hon reagerar som han/hon gör.
- Bekräfta att den andra personen har förstått vad du har sagt och tvärtom.

1.6 *Etiska regler (K2)*	*10 minuter*

Att vara programvarutestare innebär att man i bland får ta del av information av konfidentiell natur samt att man jobbar under tystnadsplikt. Detta gör att etiska principer är nödvändiga, bland annat för att säkra att information inte används på ett otillbörligt sätt. Med vedertagna etiska principer för ingenjörer från ACM och IEEE som underlag, har ISTQB® fastslagit följande etiska principer:

ALLMÄNHETEN - Certifierade programvarutestare skall handla i allmänhetens intresse.

KUND OCH ARBETSGIVARE - Certifierade programvarutestare skall agera i kunds och arbetsgivares intresse, i överensstämmelse med allmänhetens intresse.

PRODUKT - Certifierade programvarutestare skall försäkra sig om att de leverabler de tillhandahåller (för produkter och system som de testar) motsvarar högsta möjliga professionella standard.

OMDÖME - Certifierade programvarutestare ska upprätthålla integritet och oberoende sina professionella omdömen.

LEDNING - Certifierade testledare och testkoordinatorer inom programvaruutveckling skall bidra med och främja etiska tillvägagångssätt vid ledningen av programvaruprojekt.

YRKESROLL - Certifierade programvarutestare skall främja integriteten och anseendet för testyrket i överensstämmelse med allmänhetens intresse.

KOLLEGOR - Certifierade programvarutestare skall vara rättvisa mot och stötta sina kollegor samt främja samarbete med programvaruutvecklare.

SJÄLV – Certifierade testare skall delta i ett livslångt lärande avseende sitt yrkes praxis och främja ett etiskt tillvägagångssätt gentemot yrkets praxis.

Referenser

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. Testning genom programvarans livscykel (K2)	115 minuter

Inlärningsmål för testning genom programvarans livscykel

Följande målsättningar visar vad som uppnås efter avslutande av respektive modul.

2.1. Utvecklingsmodeller för programvara (K2)

IM-2.1.1. Förklara förhållandet mellan utveckling, testaktiviteter och arbetsprodukter i utvecklingens livscykel genom att ge exempel byggda på produkten och projektet (K2).
IM-2.1.2. Känna igen det faktum att utvecklingsmodeller för programvara måste anpassas till innehållet i projektet och produktens egenskaper. (K1)
IM-2.1.3. Komma ihåg egenskaper hos bra testning som är applicerbara i alla livscykelmodeller (K1).

2.2. Testnivåer (K2)

IM-2.2.1. Jämföra olika nivåer på testning:
viktiga målsättningar, typiska testobjekt, typisk inriktning (t.ex. funktionell eller strukturell) och relaterade arbetsprodukter, personer som testar, typ av fel och felsymptom som hittas. (K2)

2.3. Testtyper (K2)

IM-2.3.1. Jämföra fyra testtyper (funktionella, icke-funktionella, strukturella och ändringsrelaterade) med exempel. (K2)
IM-2.3.2. Komma ihåg att funktionella och strukturella tester förekommer på alla testnivåer. (K1).
IM-2.3.3. Identifiera och beskriva icke-funktionella testtyper baserade på icke-funktionella krav (K2).
IM-2.3.4. Identifiera och beskriva testtyper baserade på analys av programvarans systemstruktur eller arkitektur.
IM-2.3.5. Beskriva syftet med omtestning av felrättningar och regressionstestning. (K2)

2.4. Underhållstestning (K2)

IM-2.4.1. Jämföra underhållstestning (testning av ett existerande system) med testning av en ny applikation med avseende på testtyper, startvillkor för testning och omfattning av testning. (K2)
IM-2.4.2. Känna till indikatorerna för underhållstestning (modifiering, migration eller indragning). (K1)
IM-2.4.3. Beskriva betydelsen av regressionstestning och påverkansanalys i underhållsverksamheten. (K2)

2.1 *Utvecklingsmodeller för programvara (K2)*	*20 minuter*

Begrepp
COTS (Kommersiellt-Från-Hyllan), iterativ-inkrementell utvecklingsmodell, validering, verifiering, V-modell.

Bakgrund
Testning existerar inte isolerat; testaktiviteterna är kopplade till utvecklingsaktiviteterna av programvaran. Olika livscykelmodeller för utvecklingen kräver olika angreppssätt för testningen

2.1.1 V-modellen (sekventiell utvecklingsmodell) (K2)
Även om det existerar varianter på V-modellen så använder sig den normalt av fyra typer av testnivåer motsvarande de fyra utvecklingsnivåerna.

De fyra nivåerna i denna kursplan är:
- Komponent- (enhets-) testning;
- Integrationstestning;
- Systemtestning;
- Acceptanstestning.

I praktiken kan en V-modell ha fler, färre, eller andra nivåer av utveckling och testning, beroende på projektet och programvaruprodukten. Till exempel kan det finnas komponentintegrationstestning efter komponenttestning och systemintegrationstestning efter systemtestning.

Programvarans arbetsprodukter (t.ex. verksamhetsscenarier eller användningsfall, kravspecifikationer, konstruktionsdokument eller kod) som har producerats under utvecklingen utgör oftast basen för testningen på en eller flera nivåer. Referenser till generiska arbetsprodukter finns i mognadsmodeller som ”Capability Maturity Model Integration (CMMI)” eller livscykelprocesser för programvara som standarden ”Software life cycle processes” (IEEE/IEC 12207). Verifiering och validering (och tidig testdesign) kan utföras parallellt med utvecklingen av arbetsprodukter för programvaran.

2.1.2 Iterativ-inkrementell utvecklingsmodell (K2)
Iterativ-inkrementell utveckling är benämning på den process, med upprättande av krav, konstruktion, byggande och testning av ett system, som sker i många små utvecklingssteg. Exempel på detta är: att bygga en prototyp, snabbutveckling (rapid application development, RAD), Rationals utvecklingsprocess (RUP) och lättviktsmetoder för utveckling. System som produceras med hjälp av dessa modeller kan testas på flera testnivåer i varje iteration. Ett inkrement adderat till andra tidigare utvecklade inkrement, skapar ett funktionellt växande system, vilket också skall testas. Regressionstestning blir mer och mer viktigt i alla iterationer efter den första. Verifiering och validering kan utföras i varje inkrement.

2.1.3 Testning inom en livscykelmodell (K2)
I varje livscykelmodell finns det flera egenskaper som kännetecknar bra testning:
- För varje utvecklingsaktivitet finns det en motsvarande testaktivitet.
- Varje testnivå har en målsättning speciellt inriktad för denna nivå.
- Analys och konstruktion av tester för en bestämd testnivå skall starta under motsvarande utvecklingsaktivitet.
- Testarna skall medverka i granskning av dokument så fort som utkast finns tillgängliga under utvecklingens livscykel.

Beroende på karaktären hos projektet eller hos systemarkitekturen kan testnivåer kombineras och organiseras om. När man till exempel integrerar ett inköpt system eller en komponent (COTS,

kommersiellt-från-hyllan) med ett eget system så kan köparen göra integrationstestning på systemnivå (t.ex. integration med infrastrukturen och andra system) och acceptanstestning (funktionell och/eller icke-funktionell testning och användar- och/eller drifttestning).

2.2 *Testnivåer (K2)*	*40 minuter*

Begrepp
Alfatest, betatest, komponenttest, drivrutin, fälttest, funktionella krav, integration, integrationstest, icke-funktionella krav, robusthetstest, stubbe, systemtest, testdriven utveckling, testmiljö, testnivå, användaracceptanstest.

Bakgrund
För varje testnivå kan följande identifieras:
Dess generiska målsättning, arbetsprodukt(er) som refereras till för att ta fram testfall (dvs. testbasen), testobjektet (dvs. vad som skall testas), typiska fel som kommer att hittas, krav på testexekveringsplattform, verktygsstöd, specifika angreppssätt och ansvar.
Hänsyn skall även tas till testsystemets konfigurationsdata, eftersom konfigurationsdata är en del av systemet.

2.2.1 Komponenttestning (K2)
Typiska testobjekt: Komponenter, program, datakonverterings-/migreringsprogram, databasmoduler.
Testbas: Komponentkrav, detaljerad design, kod.
Komponenttestning (även känt som enhets, modul eller programtest) söker efter fel i och verifierar funktionen hos programvara (t.ex. moduler, program, objekt, klasser etc.), som är testbara separat. Det kan göras isolerat från resten av systemet, beroende på förhållandet till livscykeln för utvecklingen och till systemet. Stubbar, drivrutiner och simulatorer kan användas.

Komponenttestning kan omfatta testning av funktionalitet och specifika icke-funktionella egenskaper, som testning av resursbeteende (t.ex. minnesläckor) eller robusthet såväl som strukturerad testning (t.ex. beslutstäckning). Testfall tas fram från arbetsprodukter som en komponentspecifikation, programvarukonstruktion eller datamodellen.

Normalt sker komponenttestning med tillgång till koden som ska testas och med stöd av en utvecklingsmiljö t.ex. ramverk för enhetstestning eller avlusningsverktyg. I praktiken utförs detta av den programmerare som skrivit koden. Fel åtgärdas normalt så fort de hittas utan att de hanteras formellt.

Ett angreppssätt till komponenttestning är att förbereda och automatisera testfall innan koden skrivs. Detta angreppssätt kallas ’test-first’ eller testdriven utveckling. Det är i högsta grad iterativt och är grundat på cykler med att ta fram testfall, bygga och integrera små delar av koden, utföra komponenttester och genomföra rättningar tills testerna godkänns.
Specifika komponenter att testa inkluderar ”on-off” komponenter för konvertering/migrering av data från äldre system.

2.2.2 Integrationstestning (K2)
Typiska testobjekt: Delsystem, databasimplementation, infrastruktur, systemkonfiguration, gränssnitt och konfigurationsdata.
Testbas: Programvaru- och systemdesign, arkitektur, arbetsflöden, användningsfall.

Systemkonfiguration och konfigurationsdata bör beaktas i samband med att ett standardsystem konfigureras för ett specifikt ändamål eller miljö, t.ex. konfigurering av ett COTS-system.

Integrationstestning testar gränssnitt mellan komponenter, samspelet mellan olika delar i ett system, t.ex. operativsystem, filsystem, maskinvara eller gränssnittet mellan system.

Det kan finnas mer än en nivå av integrationstestning och den kan utföras på testobjekt av varierande storlek, till exempel:

1. Komponentintegrationstest testar samspelet mellan programvarukomponenter och görs efter komponenttestning.
2. Systemintegrationstest testar samspelet mellan olika system eller mellan hårdvara, programvara och kan göras efter systemtest. I detta fall kan utvecklingsorganisationen bara kontrollera ena sidan av samspelet så ändringar kan kanske vara destabiliserande. Verksamhetsprocesser som är realiserade som arbetsflöden kan omfatta en serie av flera system. Problem över plattformsgränserna kan bli betydande.

Ju större omfattningen på integrationen är desto svårare blir det att isolera ett fel till en specifik komponent eller ett specifikt system, vilket kan leda till en ökad risk.

Systematisk integrationsstrategi kan baseras på systemarkitekturen (t.ex. uppifrån och ned eller nedifrån och upp), funktionellt innehåll, sekvenser i en transaktionsbehandling, eller andra aspekter på systemet eller en komponent. För att underlätta att isolera och hitta defekter tidigt skall integrationen helst vara inkrementell i stället för ”big bang”.

Test av icke-funktionella egenskaper (t.ex. prestanda) kan inkluderas i integrationstestningen.

Vid varje integrationssteg skall testarna koncentrera sig enbart på själva integrationen dvs. gränssnitten. Vid till exempel integration av modul A med modul B skall de fokusera på kommunikationen mellan modulerna och inte funktionaliteten i de individuella modulerna, som gjordes i komponenttest. Angreppssättet kan vara både funktionellt och strukturellt.

Idealt är om testarna förstår arkitekturen och påverkar integrationsplaneringen. Om integrationstesterna planeras före byggandet av komponenter eller system kan komponenterna konstrueras i den ordning som ger mest effektiv testning.

2.2.3 Systemtestning (K2)

Typiska testobjekt: Systemet, manualer, systemkonfiguration och konfigurationsdata.
Testbas: Kravspecifikation för system och programvara, funktionsspecifikation, riskanalys.

Systemkonfiguration och konfigurationsdata bör beaktas i samband med att ett standardsystem konfigureras för ett specifikt ändamål eller miljö, t.ex. konfigurering av ett COTS-system.

Systemtestning täcker hela systemets/produktens beteende. Testningens omfattning skall vara tydligt beskriven i den övergripande testplanen (master test plan) eller i testplanen för aktuell nivå. Testmiljön skall, för systemtest, efterlikna den slutliga mål- eller produktionsmiljön så mycket som möjligt för att minimera risken för att fel kopplade till miljön inte hittas under testningen.

Systemtest kan omfatta testning baserat på risker och/eller kravspecifikationer, verksamhetsprocesser eller användningsfall. Den kan också baseras på andra högnivåbeskrivningar av systemets beteende eller samverkan med operativsystem och systemresurser.

Systemtest kan undersöka funktionella och icke-funktionella krav på systemet samt datakvalitetsegenskaper. Krav kan finnas i textform och/eller modeller. Testare skall också kunna hantera ofullständiga och odokumenterade krav. Systemtestning av funktionella krav startar med att använda mest lämpliga specifikationsbaserade (black-box) tekniker för att testa den funktionella aspekten på systemet. En beslutstabell kan till exempel skapas, innehållande kombinationer av effekter som beskrivs i verksamhetsreglerna. Strukturbaserade tekniker (white-box) kan sedan användas för att utvärdera grundligheten i testningen med avseende på ett strukturellt element som t.ex. strukturen hos en meny eller navigeringen av en websida (Se kapitel 4).

Systemtest utförs oftast av ett oberoende testteam.

2.2.4 Acceptanstestning (K2)

Typiska testobjekt: Affärsprocesser för helt integrerade system, drift- och underhållsprocesser, användarförfaranden, formulär, rapporter, konfigurationer och konfigurationsdata.
Testbas: Användarkrav, systemkrav, användningsfall, affärsprocesser, riskanalys.

Oftast är det kunderna eller andvändarna som har ansvar för acceptanstestning, men även andra intressenter kan vara inblandade.

Målet för acceptanstestning är att skapa en tilltro till systemet, till delar av det, eller till specifika icke-funktionella egenskaper av systemet. Att hitta fel är inte det viktigaste målet i acceptanstestning. Acceptanstestning kan utvärdera systemets status inför driftsättning och användning även om det inte är den sista nivån i testningen. En systemintegrationstest kan till exempel följa efter acceptanstest av ett system.

Acceptanstestning kan förekomma på fler ställen i programvarans livscykel, till exempel:

- En inköpt programvaruprodukt (t.ex. COTS) kan bli acceptanstestad när den installeras eller integreras.
- Acceptanstestning av användningen av en komponent kan utföras under komponenttestning.
- Acceptanstestning av nya funktionella förbättringar kan göras före systemtestning.

Representativa sätt för acceptanstestning kan vara följande:

Användaracceptanstestning
Utföres normalt av verksamhetsanvändarna och verifierar om systemet är klart för användning.

Acceptanstestning för drift
Acceptanstestning som utförs av systemadministratörer:

- Testning av funktioner för tagning av säkerhetskopia med återställning;
- Återhämtning efter katastrof;
- Användarhantering;
- Underhållsarbete;
- Datainmatning och migreringsaktiviteter;
- Periodisk kontroll av sårbarheten med avseende på säkerhet.

Acceptanstest av kontrakt och förordningar
Acceptanstestning av kontrakt görs mot ett kontrakts acceptanskriterier för framtagning av kundutvecklad programvara. Acceptanskriterierna skall definieras vid kontraktsöverenskommelsen. Acceptanstestning av förordningar görs mot alla typer av förordningar som formellt måste följas, t.ex. statliga, lagliga eller säkerhetskritiska reglementen.

Alfatestning och beta- (eller fält-) testning
Utvecklare av marknadsprogramvara eller COTS vill ofta ha återkoppling från potentiella eller existerande kunder innan programvaran ges ut för kommersiell försäljning. Alfatestning utförs hos utvecklingsföretaget, men inte av utvecklingsteamet, medan betatestning eller fälttestning utförs av kunder eller potentiella kunder i sina egna miljöer.
Organisationer kan använda andra begrepp eller termer än de som förekommit här såsom t.ex. fabriksacceptanstestning och platsacceptanstestning för system som har testats före och efter det att de flyttats till kundens plats.

2.3 *Testtyper (K2)*	*40 minuter*

Begrepp
Black-box-test, kodtäckning, bekräftelsetest, funktionell test, interoperatibilitetstest, lasttest, underhållstestning, prestandatest, portabilitetstest, tillförlitlighetstest, säkerhetstest, stresstest, strukturell testning, användbarhetstest, white-box-test.

Bakgrund
En grupp av testaktiviteter kan syfta till att verifiera ett programvarusystem (eller del av ett system) baserat på ett specifikt testskäl eller testmål.

En testtyp fokuserar på ett särskilt testmål, till exempel:
- Test av en funktion hos programvaran;
- Test av en icke-funktionell kvalitetsegenskap som tillförlitlighet eller användbarhet, struktur eller arkitektur hos komponent, programvara eller system.
- Test kopplad till ändringar, dvs. bekräfta att ett fel har åtgärdats (omtestning) och visar om oavsiktliga förändringar har uppstått (regressionstestning).

En modell av programvaran kan utvecklas och/eller användas i strukturell testning (t.ex. en kontrollflödesmodell eller en menystrukturmodell), ickefunktionell testning (t.ex. en prestandamodell, användbarhetsmodell eller en säkerhetshotsmodell) och funktionell testning (t.ex. en processflödesmodell, en tillståndsövergångsmodell eller en ren textspecifikation).

2.3.1 Testning av funktionalitet (funktionell testning) (K2)
De funktioner som ett system, delsystem eller komponent skall utföra kan beskrivas i arbetsprodukter såsom kravspecifikation, användningsfall, en funktionsspecifikation, eller de kan vara odokumenterade. Funktioner visar ”vad” systemet gör.

Funktionella tester är baserade på funktioner och finesser (beskrivna i dokument eller förstådda av testarna) och dess interoperatibilitet med specifika system, och kan utföras på alla testnivåer (t.ex. kan testning av komponenter vara baserade på en komponentspecifikation)

Specifikationsbaserade tekniker kan användas för att ta fram testvillkor och testfall för programvarans eller systemets funktionalitet (se kapitel 4). Testning av funktionalitet behandlar det externa beteendet hos programvaran (black-box testing).

En typ av funktionell testning, säkerhetstestning, undersöker de funktioner (t.ex. en brandvägg) som är relaterade till upptäckande av hot, t.ex. virus, från utomstående med uppsåt att skada. En annan typ av funktionell testning, interoperatibilitetstestning, utvärderar programvarans förmåga att interagera med en eller flera andra specificerade komponenter eller system.

2.3.2 Testning av icke-funktionella programvaruegenskaper (icke-funktionell testning) (K2)
Icke-funktionell testning inkluderar, men är inte begränsat till testning av prestanda, last, användbarhet, underhållbarhet, tillförlitlighet och portabilitet. Det kan ses som testning av ”hur” systemet fungerar.

Icke-funktionell testning kan förekomma på alla testnivåer. Termen icke-funktionell testning beskriver de tester som krävs för att mäta egenskaperna hos ett system eller en programvara och som kan kvantifieras på en varierande skala, t.ex. svarstider vid prestandatestning. Dessa tester kan refereras till en kvalitetsmodell, som den definierad av standarden för programkvalitet -‘Software Engineering – Software Product Quality’ (ISO 9126). Icke funktionell testning fokuserar på systemets externa beteende och oftast används black-box testning för att åstadkomma detta.

2.3.3 Testning av programvarans struktur/arkitektur (strukturell testning) (K2)

Strukturell (white-box) testning kan utföras på alla testnivåer. Strukturella tekniker används bäst efter specifikationsbaserad tekniker för att hjälpa till att mäta grundligheten hos testningen genom att utvärdera täckningen för en viss strukturtyp.

Täckning är den grad till vilken en struktur har blivit utförd av en testsvit, uttryckt som en procentsats av de delar som blivit täckta. Om täckningen inte är 100 % kan det tänkas att fler tester ska konstrueras för att testa de delar som hoppats över och på så sätt öka täckningsgraden. Tekniker för testtäckning beskrivs i kapitel 4.

På alla testnivåer, men speciellt inom komponenttestning och komponentintegrationstestning, kan verktyg användas för att mäta kodtäckning hos beståndsdelarna, som t.ex. kodsatser eller beslut. Strukturell testning kan även baseras på systemets arkitektur, t.ex. anropshierarki.

Angreppssätt för strukturell testning kan användas i systemtestning, systemintegrationstestning eller acceptanstestning (t.ex. verksamhetsmodeller eller menystrukturer).

2.3.4 Testning vid ändringar: Omtest och regressionstestning) (K2)

När ett fel har upptäckts och åtgärdats skall programvaran omtestas för att bekräfta att urspungsfelet verkligen har försvunnit. Detta kallas omtestning. Avlusning (lokalisering och åtgärdande av fel) är en aktivitet för utvecklingen, inte testningen.

Regressionstestning är en upprepad test av ett tidigare testat program eller system, efter en ändring, för att upptäcka fel som introducerats eller gjorts synliga som en bieffekt av ändringarna. Dessa fel kan antingen finnas i den programvara som testas, eller i en annan beroende eller oberoende programvarukomponent. Regressionstestning utförs när programvaran eller dess miljö har förändrats. Till vilken omfattning regressionstestningen skall göras beror på sannolikheten att inte hitta fel i den programvara som tidigare fungerade.

Testfall skall vara repeterbara om de skall användas för omtestning och för att stödja regressionstestning.

Regressionstestning kan utföras på alla testnivåer och kan användas vid funktionell, icke-funktionell och strukturell testning. Testsviter för regressionstest körs många gånger och växer normalt långsamt fram; därför är regressionstestning en stark kandidat för automatisering.

2.4 *Underhållstestning (K2)*	*15 minuter*

Begrepp
Påverkansanalys, underhållstestning.

Bakgrund
När ett programvarusystem en gång har blivit driftsatt kan det vara i drift i åratal. Under denna tid har ofta systemet, dess konfiguration och dess miljö rättats, ändrats eller utökats. Att planera frisläpp i förväg är avgörande för en framgångsrik underhållstest. Det är viktigt att skilja på planerade frisläpp och 'hot-fixar'.
Underhållstestning görs på ett befintligt operativsystem som plattform och initieras på grund av ändringar, migration (t.ex. flytt till en annan plattform) eller indragning av programvaran eller systemet.

Ändringar inkluderar planerade förbättringar (t.ex. utgåvebaserade), korrigerande och akuta rättningar eller ändringar i miljön. De kan vara planerade uppgraderingar av operativsystem, databaser, eller COTS (Kommersiellt-Från-Hyllan) program eller fixar för nya synliga eller nya upptäckta svagheter i operativsystemet.

Underhållstestning vid migrering (t.ex. från en plattform till en annan) skall omfatta, förutom driftstestning i den nya miljön, också testning av den förändrade programvaran. Migrationstestning (konverteringstestning) måste också utföras om, till exempel, data från någon annan applikation behöver migreras till det underhållna systemet.

Underhållstestning vid indragning av ett system kan omfatta testning av datamigrering eller arkivering om det finns ett krav att bevara data en längre tid.

Förutom testning av vad som har förändrats så inkluderar underhållstestningen också regressions-testning av de delar av systemet som inte har ändrats. Omfattningen på underhållstestningen är beroende av risker kopplade till ändringar, storleken på det befintliga systemet och av storleken på ändringen. Beroende på vad som ändrats så kan underhållstestningen utföras på alla testnivåer och omfatta alla testtyper.

Att avgöra hur det befintliga systemet påverkas av ändringar kallas påverkansanalys. Detta är en hjälp till att besluta om hur mycket regressionstestning som skall göras.

Underhållstestning kan vara svårt att utföra om specifikationer är inaktuella eller saknas, eller om testare med domänkunskap saknas.

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

3. **Statiska tekniker (K2)**	**60 minuter**

Inlärningsmål för statiska tekniker

Följande målsättningar visar vad som uppnås efter avslutande av respektive modul.

3.1. Statiska tekniker och testprocessen (K2)

IM-3.1.1. Känna till vilka arbetsprodukter inom programvaruutvecklingen som kan granskas med hjälp av olika statiska tekniker. (K1)
IM-3.1.2. Förklara värdet och vikten av användandet av statiska tekniker för utvärdering av arbetsprodukter inom programvaruutvecklingen. (K2)
IM-3.1.3. Förklara skillnaden mellan statiska och dynamiska tekniker med avseende på mål, typer av defekter som hittas och dessa teknikers roller i programvarans livscykel. (K2)

3.2. Granskningsprocessen (K2)

IM-3.2.1. Komma ihåg de aktiviteter, roller och ansvar som gäller för en formell granskning (K1)
IM-3.2.2. Förklara skillnader mellan olika typer av granskningar: informell granskning, teknisk granskning, genomgång och inspektion. (K2)
IM-3.2.3. Förklara framgångsfaktorer för ett lyckat genomförande av granskningar. (K2)

3.3. Statisk analys med verktyg (K2)

IM-3.3.1. Komma ihåg vilka typiska defekter och fel som kan upptäckas med statisk analys och jämföra dem med vad som kan hittas av granskningar och dynamisk testning. (K1)
IM-3.3.2. Beskriv, med hjälp av exempel, typiska fördelar med statisk analys. (K2)
IM-3.3.3. Lista kod- och designdefekter som kan upptäckas av ett statiskt analysverktyg. (K1)

3.1 *Statiska tekniker och testprocessen (K2)*	*15 minuter*

Begrepp
Dynamisk testning, statisk testning.

Bakgrund
Till skillnad mot dynamisk testning, som kräver att programvaran exekveras, så förlitar sig statiska testtekniker på manuell kontroll (granskning) och automatiserad analys (statisk analys) av källkoden eller andra producerade dokument utan att programvaran exekveras.

Granskningar är ett sätt att testa arbetsprodukter (inklusive koden) och kan med fördel utföras före dynamisk testning. Defekter som upptäcks genom granskningar i ett tidigt skede (t ex granskningar av krav) är oftast mycket billigare att rätta än de defekter som hittas när man utför dynamisk testning av redan utvecklad kod.

Granskningar kan utföras helt manuellt, men det finns också verktygsstöd för detta. Den huvudsakliga manuella aktiviteten är att granska arbetsprodukten av ett arbete och kommentera de avvikelser man hittar. Alla arbetsprodukter kan granskas, inklusive kravspecifikationer, designspecifikationer, kod, testplaner, testspecifikationer, testfall, testskript, användarmanualer eller webbsidor.

Fördelarna med granskningar är tidig fellokalisering och rättning, ökad produktivitet inom utveckling, minskad utvecklingstid, minskad tid och kostnad för testning, minskade produktlivstidskostnader, färre fel i senare faser och förbättrad kommunikation. Vid granskningar kan ofullständigheter upptäckas, t.ex. att krav saknas i en specifikation, vilket är svårt i dynamisk testning.

Granskningar, statisk analys och dynamisk testning har samma övergripande mål – påvisa fel. De kompletterar varandra: de olika teknikerna ger möjlighet att effektivt och ändamålsenligt hitta olika typer av fel. I motsats till dynamisk testning hittar statiska tekniker fel (defekter) snarare än själva felsymptomen.

Typiska defekter som är lättare att hitta med granskningar än med dynamisk testning innefattar: avvikelser från standard, kravdefekter, designdefekter, otillräcklig underhållbarhet och felaktigt specificerade gränssnitt.

3.2 *Granskningsprocess (K2)*	*25 minuter*

Begrepp
Startkriterier, formell granskning, informell granskning, inspektion, mätetal, moderator, kollegial granskning, granskare, skribent, teknisk granskning, genomgång.

Bakgrund
De olika typerna av granskningar varierar från att vara informella, som kännetecknas av avsaknad av skrivna instruktioner till granskare, till systematiska, som kännetecknas av delaktighet av teamet, dokumenterade granskningsresultat och dokumenterad granskningsprocess. Formaliteten hos en granskningsprocess är relaterad till faktorer såsom mogenhetsgraden i utvecklingsprocessen, varje legalt eller regulatoriskt krav eller behovet av uppföljning.

Hur en granskning genomförs är beroende av de överenskomna målen för granskningen (t.ex. hitta defekter, vinna samförstånd, utbilda testare och nya teammedlemar, eller diskussion och beslut med konsensus).

3.2.1 Aktiviteter i en formell granskning (K1)
En typisk formell granskning innehåller följande huvudsteg:

1. Planering:
    - Definiera granskningskriterier
    - Välja personal
    - Tilldela roller
    - Sätta upp start- och avslutskriterier för mer formella granskningstyper (såsom inspektioner):
    - Välja vilka delar av dokumenten som skall granskas.
    - Stämma av startkriterier (vid mer formella granskningstyper).
2. Start:
    - Dela ut dokument
    - Förklara målsättningen, processen och dokumenten som skall granskas för alla deltagare.
3. Individuella förberedelser:
    - Arbete som genomförs enskilt av varje deltagare före granskningsmötet.
    - Notera möjliga defekter, frågor och kommentarer.
4. Genomgång/utvärdering/dokumentering av resultat (Granskningsmöte):
    - Diskussioner eller loggning, med dokumenterade resultat eller med mötesprotokoll (vid mer formella granskningstyper).
    - Mötesdeltagarna kan helt enkelt notera defekterna, ge rekommendationer för hur defekterna skall hanteras eller fatta beslut om defekterna.
    - Genomgång/utvärdering och dokumentering av frågor kan ske på fysiska möten eller genom elektroniskt konferensmöte.
5. Omarbete:
    - Åtgärda de funna defekterna, utförs oftast av författaren.
    - Uppdatera och dokumentera status på defekterna (vid formella granskningar)
6. Uppföljning:
    - Kontrollera att alla defekter har blivit tilldelade för åtgärd
    - Sammanställa mätetal
    - Kontrollera att avslutskriterier är uppnådda (vid mer formella granskningstyper).

3.2.2 Roller och ansvar (K1)
En typisk formell granskning innehåller följande roller:

- Ledare/chef: bestämmer över genomförandet av granskningar, avsätter tid i projektplaner och avgör om målsättningen för granskningarna blivit uppfyllda.

- Moderator: den person som leder granskningen av dokumentet eller uppsättningen av dokument. Det inkluderar planering av granskningen, hålla mötet, följa upp efter mötet och om nödvändigt medla mellan olika åsikter. Moderatorn är oftast den person som ligger bakom en lyckad granskning.
- Författare: eller den person som ansvarar för dokumentet eller dokumenten som skall granskas.
- Granskare: individer med en specifik teknisk eller verksamhetsbakgrund som, efter de nödvändiga förberedelserna, identifierar och beskriver t.ex. defekterna i det som granskats. Val av granskare skall göras så att de representerar olika områden och roller i granskningsprocessen och skall delta i granskningsmöten.
- Sekreterare (eller skribent): dokumenterar allt som kommer fram, problem och öppna frågor som identifieras under mötet.

Att granska programvara och andra arbetsprodukter från olika perspektiv och att använda sig av checklistor, kan göra granskningar mer effektiva och ändamålsenliga. Exempel kan vara att granska efter en checklista som baseras på vyer från användare, underhållspersonal, testare, driftpersonal, eller en checklista för typiska kravproblem.

3.2.3 Granskningstyper (K2)

En programvara eller arbetsprodukt kan bli föremål för mer än en granskning. Om mer än en typ av granskning används, kan ordningen variera. Till exempel kan en informell granskning kanske utföras före en teknisk granskning, eller en inspektion av en kravspecifikation kanske utförs före en genomgång med kunderna. Huvudsakliga kännetecken, val och syften med vanliga granskningstyper är:

Informell granskning
Viktiga kännetecken:
- ingen formell process;
- kan ske genom parprogrammering eller att en teknisk ledare leder design- och kod-granskningen;
- frivilligt dokumenterad;
- nyttan av granskningen kan variera beroende på granskaren;
- huvudsyfte: att på billigaste sätt att uppnå en viss nytta.

Genomgång
Viktiga kännetecken:
- mötet leds av författaren;
- kan ske med hjälp av scenarion, torrsimulering, grupp med kolleger;
- ej tidsbegränsade sammanträden;
- följande kan vara valfritt: granskarna är förberedda, granskningsrapport skrivs, avvikelselista tas fram och sekreterare (som inte är författaren) utses;
- kan variera i utförande från ganska informellt till mycket formellt;
- huvudsyfte: lärande, ge förståelse, hitta defekter.

Teknisk granskning
Viktiga kännetecken:
- dokumenterad, definierad defektutpekningsprocess som inkluderar kolleger och tekniska experter, ledningens deltagande är valfritt;
- kan utföras som en granskning med kolleger utan ledarens/chefens deltagande;
- önskvärt att granskningen leds av en utbildad moderator (inte författaren);
- förberedelse innan mötet;
- valfritt användande av checklistor;
- granskningsrapport som inkluderar defektlista, beslut om programvaran lever upp till kraven och, där det är möjligt, rekommenderade åtgärder för defekterna;

- kan variera i utförande från ganska informellt till mycket formellt;
- huvudsyfte: diskussioner, beslutsfattande, utvärdering av alternativ, hitta defekter, lösa tekniska problem och kontroll av att det granskade stämmer överens med specifikationer, planer, regelverk och standarder.

Inspektion
Viktiga kännetecken:

- leds av en utbildad moderator (inte av författaren);
- vanligtvis genomförd av jämlika kolleger;
- definierade roller;
- inkluderar mätetal;
- formell process som är baserad på regler och checklistor;
- definierade start- och avslutskriterier för att acceptera programvaran;
- förberedelser innan mötet;
- inspektionsrapport, lista över iakttagelser;
- formell uppföljningsprocess; (valfritt med processförbättringsprocess)
- valfritt med uppläsare;
- huvudsyfte: hitta defekter.

Genomgångar, tekniska granskningar och inspektioner kan utföras inom en kollegial grupp – kollegor på samma avdelning inom en organisation. Denna typ av granskning kallas en ”kollegial granskning”.

3.2.4 Framgångsfaktorer vid granskningar (K2)

Framgångsfaktorer hos granskningar är:

- Varje granskning har en klar fördefinierad målsättning.
- Rätt personer för de uppsatta granskningsmålen är inblandade.
- Testare är värdefulla granskare som aktivt bidrar till att höja granskningskvaliteten och som genom sitt deltagande lär sig produkten och därför kan förbereda testningen tidigare.
- Defekter som hittas tas väl emot och uttrycks objektivt.
- Mjuka faktorer och psykologiska aspekter måste hanteras (t.ex. att göra granskningen till en positiv erfarenhet för författaren).
- Förtroende: resultatet av granskningen används inte till att utvärdera deltagarna.
- Granskningsteknikerna anpassas så att de passar programvaran och granskarna och för att uppnå granskningens målsättning.
- Om det är tillämpligt så används checklistor eller roller för att öka effektiviteten i att hitta defekter.
- Att utbildning ges i granskningstekniker, speciellt i de mer formella teknikerna, såsom inspektioner.
- Ledningen stödjer en bra granskningsprocess (t ex genom att avsätta tillräckligt med tid för granskningsaktiviteter i projekttidsplaner).
- Det finns en tonvikt på lärande och processförbättringar.

3.3 *Statisk analys med verktyg (K2)*	*20 minuter*

Begrepp
Kompilator, komplexitet, kontrollflöde, dataflöde, statisk analys.

Bakgrund
Målsättningen med statisk analys är att hitta defekter i källkod och programvarumodeller. Statisk analys utförs genom granskning med hjälp av ett verktyg utan att programvaran exekveras; dynamisk testning exekverar programkoden. Statisk analys kan lokalisera defekter som är svåra att upptäcka med dynamisk testning. Precis som vid granskning hittar statisk analys defekter snarare än felyttringar. Statiska analysverktyg analyserar programkoden (t ex kontrollflöde och dataflöde), såväl som det utdata som genereras t.ex. HTML och XML.

Fördelen med statisk analys är:
- Tidig identifiering av defekter innan testexekvering.
- Tidig varning om tveksam utformning av kod eller design, genom beräkning av mätetal, såsom hög komplexitet.
- Identifiering av defekter som inte är lätta att hitta genom dynamisk testning.
- Upptäckt av beroenden och motsättningar i programvarumodeller, såsom länkar.
- Förbättrat underhåll av kod och design.
- Förebyggande av defekter, om utvecklarna lär sig av sina misstag.

Typiska defekter som kan hittas av ett statiskt analysverktyg är:
- referens till en variabel med ett icke definierat värde;
- inkonsekvent gränssnitt mellan moduler och komponenter;
- variabler som inte används eller är felaktigt deklarerade;
- kod som aldrig används, så kallad död kod;
- saknad eller felaktig logik (eventuellt oändliga loopar)
- Onödigt komplicerade kodkonstruktioner
- avvikelser mot programmeringsregler;
- säkerhetssårbarheter;
- syntaxfel i kod och programvarumodeller.

Statiska analysverktyg används oftast av utvecklare (granskning mot fördefinierade normer eller programmeringsregler) före och under komponent- och integrationstestning eller när koden checkas in i konfigurationshanteringsverktyg och av utvecklare under programvarumodellering. Statiska analysverktyg kan skapa ett stort antal varningsmeddelanden, som måste hanteras för att nå ett effektivt utnyttjande av verktyget.

Kompilatorer kan erbjuda ett visst stöd för statisk analys och beräkningar av mätetal.

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

4. Testdesigntekniker (K4) — 285 minuter

Inlärningsmål för tekniker för testdesigntekniker

Följande målsättningar visar vad som uppnås efter avslutande av respektive modul.

4.1. Testutvecklingsprocessen (K3)

IM-4.1.1. Skilja mellan specifikationer för testdesign, testfall respektive testinstruktion. (K2)
IM-4.1.2. Jämföra begreppen testvillkor, testfall och testinstruktion. (K2)
IM-4.1.3. Utvärdera kvaliteten hos testfall. Klarar testfallen att:
- tydligt visa spårbarhet mot kraven,
- ange förväntat resultat. (K2)

IM-4.1.4. Tolka testfall för att skriva välstrukturerade testinstruktioner på en detaljnivå som är relevant för målgruppen, dvs. de testare som skall använda instruktionerna. (K3)

4.2. Kategorisering av tekniker för testdesign (K2)

IM-4.2.1. Minnas att orsaker till att både specifikationsbaserade (black-box) och strukturbaserade (white-box) testdesigntekniker är användbara och nämna vanliga exempel av vardera typen. (K1)
IM-4.2.2. Beskriva utmärkande egenskaper, likheter och skillnader mellan, tekniker som är baserade på specifikation, på struktur och på erfarenhet. (K2)

4.3. Specifikationsbaserade tekniker (black-box) (K3)

IM-4.3.1. Skriva testfall utgående från givna programmodeller genom att använda de följande testdesigntekni kerna: (K3)
- ekvivalensuppdelning,
- gränsvärdesanalys,
- testning med hjälp av beslutstabeller,
- tillståndsbaserad testning.

IM-4.3.2. Förklara huvudorsaken till vardera av de fyra teknikerna och vilken testnivå och testtyp som respektive teknik används för, samt hur täckning kan mätas. (K2)
IM-4.3.3. Förklara begreppet användarscenario och dess fördelar. (K2)

4.4. Strukturbaserade tekniker (white-box-test) (K4)

IM-4.4.1. Beskriva begreppet och värdet av kodtäckning. (K2)
IM-4.4.2. Förklara begreppen kodsatstäckning och beslutstäckning och motivera varför dessa begrepp även kan användas på andra testnivåer än komponenttest, till exempel på verksamhetsprocedurer på systemnivå. (K2)
IM-4.4.3. Skriva testfall genom att utgå från kontrollflöde och använda de följande designteknikerna: (K3)
- kodsatstestning,
- beslutstestning

IM-4.4.4. Utvärdera uppnådd kodsats- och beslutstäckning mot definierade avslutskriterier. (K4)

4.5. Erfarenhetsbaserade tekniker (K2)

IM-4.5.1. Minnas orsaker för att skriva testfall baserat på intuition, erfarenhet och kunskap om vanliga fel. (K1)
IM-4.5.2. Jämföra erfarenhetsbaserade med specifikationsbaserade testtekniker. (K2)

4.6. Välja testtekniker (K2)

IM-4.6.1. Klassificera testdesignteknikerna med avseende på deras lämplighet: att användas i ett givet sammanhang; för testbasen; utifrån givna modeller samt att testa olika programvaruegenskaper. (K2)

4.1 *Testutvecklingsprocessen (K2)*	*15 minuter*

Begrepp
Testfallsspecifikation, testdesign, testexekveringsschema, testinstruktionsspecifikation, testskript, spårbarhet.

Bakgrund
Testutvecklingsprocessen som beskrivs i detta kapitel kan utföras på olika sätt, allt från mycket informella sätt med knappt någon, eller till och med utan, dokumentation till fullt formella sätt (så som det beskrivs nedan). Vald nivå beror på sammanhanget, till exempel organisationen och utvecklingsprocessens mognadsgrad, tidsgränser, säkerhets- eller reglerade krav och inblandad personal.

Under testanalysen analyseras testbasens dokumentation för att se vad som skall testas, det vill säga identifiera testvillkoren. Ett sådant villkor definieras som ett objekt eller en händelse som kan verifieras av ett eller flera testfall, exempelvis en funktion, en tillståndsövergång, en kvalitetsegenskap eller ett strukturellt element.

Genom att säkerställa spårbarhet från testvillkoren tillbaka till specifikationer och krav blir det möjligt att både göra en effektiv konsekvensanalys vid ändrade krav och att fastställa kravtäckning för en viss mängd tester. Under testanalysen används ett detaljerat angreppssätt för att välja den testdesignteknik som kommer att användas och vara baserat på bland annat identifierade risker. (Se kapitel 5 om riskanalys)

Under testdesignen specificeras och skapas testfall och testdata . Ett testfall består av en en uppsättning indata, förutsättningar för utförande, förväntat resultat och sluttillstånd som resultat av exekveringen, framtaget för att täcka ett visst testmål eller testvillkor. I standarden IEEE STD 829-1998 (Standard for Software Test Documentation) beskrivs vad specifikationer för testdesign (med testvillkor inkluderat) respektive testfall skall innehålla.

Förväntat resultat bör tas fram som en del av beskrivningen av ett testfall och inkluderar utdata, ändringar i data och tillstånd och även andra resultat från testfallet. Om inget förväntat resultat har beskrivits, så finns risken att ett felaktigt antagande som verkar vara rimligt kan tolkas som ett korrekt resultat. Förväntat resultat bör helst beskrivas innan testet genomförs.

Vid testimplementering utvecklas, implementeras, prioriteras och organiseras testfallen i en testinstruktion (IEEE STD 829-1998). Instruktionen beskriver en sekvens av åtgärder för genomförandet av ett test. Om testet utförs med hjälp av ett testexekveringsverktyg måste sekvensen vara specificerad i ett testskript, med andra ord en automatisk testinstruktion.

De olika testinstruktionerna och automatiserade testskripten är därefter sammanställt i testexekveringsschema som definierar vilka testinstruktioner och testskript som skall utföras. Schemat tar hänsyn till faktorer som regressionstester, prioriteringar samt tekniska och logiska beroenden.

4.2 *Kategorisering av tekniker för testdesign (K2)*	*15 minuter*

Begrepp
Black-box-testdesigntekniker, erfarenhetsbaserade testdesigntekniker, testdesigntekniker, white-box-testdesigntekniker.

Bakgrund
Målet med tekniker för testdesign är att identifiera och skapa testvillkor, testfall och testdata.

Historiskt sett delas testdesigntekniker in i black-box- eller white-box tekniker. Black-box tekniker för testdesign (även kallat specifikationsbaserade tekniker) är ett sätt att härleda testvillkor, testfall och testdata genom att analysera dokumentationsdelen av testbasen. Detta gäller både funktionell och icke-funktionell testning. Black-box testning använder, per definition, ingen information gällande den interna strukturen av komponenten eller systemet som ska testas. Med hjälp av white-box tekniker för testdesign (även kallat strukturbaserade tekniker) görs motsvarande urval genom att analysera strukturen hos komponenten eller systemet. Genom att använda erfarenhetsbaserade testtekniker, i kombination med black-box och white-box tekniker, kan även utvecklares och testares erfarenhet användas till att påverka vad som skall testas.

En del tekniker tillhör enbart ena sorten, andra tekniker har delar från båda.

I denna kursplan refereras specifikationsbaserade angreppssätt till black-box-tekniker samt strukturbaserade som white-box tekniker för testdesign.

Gemensamma egenskaper för specifikationsbaserade tekniker för testdesign är:
- Modeller används för att beskriva problemet som skall lösas, program eller dess komponenter. De kan vara formella eller informella.
- Från dessa modeller härleds testfallen systematiskt.

Gemensamma egenskaper för strukturbaserade tekniker för testdesign är:
- Kunskap om hur systemet är uppbyggt, exempelvis kod och design, används för att härleda testfall.
- Det går att mäta hur stor andel av ett program som täcks av testfall och fler testfall kan skrivas för att systematiskt öka täckningsgraden.

Gemensamma egenskaper för erfarenhetsbaserade tekniker för testdesign är:
- Kunskap och erfarenhet hos individer används för att härleda testfall:
- En källa till information är kunskap om programvaran, dess användning och dess miljö hos testare, utvecklare, användare eller andra intressenter.
- Kunskap om vanliga och sannolika fel och deras fördelning är en annan källa till information.

4.3 *Specifikationsbaserade tekniker (black-box) (K3)*	*150 minuter*

Begrepp
Gränsvärdesanalys, testning med hjälp av beslutstabeller, ekvivalensklasser, tillståndsbaserad testning, användningsfallsbaserad testning.

4.3.1 Ekvivalensklassindelning (K3)

Vid ekvivalensklassindelning delas systemets eller programmets indata upp i grupper eller klasser där indata i varje grupp kommer att uppvisa likartat beteende, det vill säga de kommer sannolikt att hanteras på samma sätt. Ekvivalensklasser, som är resultatet av indelningen, kan identifieras för indata som är både giltigt (dvs. giltiga värden som skall accepteras) och ogiltigt (dvs. för ogiltiga värden som skall avfärdas). Klasser kan också identifieras även för utdata, interna värden, tidsberoende värden (exempelvis före eller efter en händelse) samt för gränssnittsparametrar (nyttiga vid integrationstest). Tester kan skapas för att täcka alla giltiga och ogiltiga ekvivalensklasser. Ekvivalensklassindelning är användbart på alla testnivåer.

Tekniken kan användas för att uppnå målen för in- och utdatatäckning. Den kan användas för manuell inmatning, för indata via olika tekniska gränssnitt till system eller för gränssnittsparametrar vid integrationstestning.

4.3.2 Gränsvärdesanalys (K3)

Beteendet vid gränserna av varje ekvivalensklass uppvisar en större risk för fel än beteendet för värden inom ekvivalensklassen, därför är gränsvärdestestning ett fruktbart sätt att upptäcka fel. Gränserna består av det minimala och det maximala värdet inom en klass. Ett gränsvärde för en giltig klass kallas för giltigt gränsvärde, ett gränsvärde för en ogiltig klass för ogiltigt gränsvärde. Det går att konstruera tester som täcker både giltiga och ogiltiga gränsvärden. Vid designen av ett testfall, väljs ett test för varje gränsvärde.

Gränsvärdesanalys kan användas på alla testnivåer. Den är relativt enkel att använda och dess felupptäckningsförmåga är hög. Detaljerade specifikationer är till stor hjälp för att kunna identifiera intressanta gränsvärden.

Denna teknik ses ofta som en utökning av ekvivalensklassindelning eller annnan black-box teknik för testdesign. Gränsvärdesanalys kan användas vid många olika testproblem t.ex. ekvivalensklasser för inmatningsfält, tids- eller tabellgränser. Gränsvärden kan även användas för testdataurval.

4.3.3 Testning med hjälp av beslutstabeller (K3)

Beslutstabeller är ett beprövat sätt att fånga systemkrav som består av logiska beroenden och att dokumentera ett systems interna uppbyggnad. Tabellerna kan användas för att dokumentera komplex affärslogik som ett system skall implementera. För att skapa beslutstabellerna analyseras specifikationerna så att systembeteenden och de villkor som påverkar dessa beteenden kan identifieras.

Systembeteenden och de indatavillkor som påverkar dessa beteenden uttrycks ofta som sant eller falskt (Boolskt). En beslutstabell innehåller de utlösande villkoren, ofta i kombination med sant och falsktvärden för all indata och tillsammans med den resulterande händelsen. Varje kolumn i tabellen motsvarar en regel i affärslogiken som beskriver vilken unik kombination av förhållanden som skall resultera i vilka händelser. Ofta beräknas täckningen genom att ha minst ett test per kolumn i tabellen, något som normalt innebär att alla kombinationer av utlösande villkor täcks.

Styrkan med att testa med hjälp av beslutstabeller ligger i att det skapas villkorskombinationer som annars inte hade utförts under test. Det kan användas i alla lägen där programbeteende beror på flera logiska val.

4.3.4 Tillståndsbaserad testning (K3)

Ett system kan visa olika svar eller resultat beroende på vilka villkor och vilket tidigare händelseförlopp (dess olika tillstånd) som skett. I dessa fall kan systemet visas som ett tillståndsdiagram. Det tillåter testare att se programmet eller systemet i termer av tillstånd, övergångar mellan tillstånden, invärden eller händelser som gör att tillstånden ändras, dvs. en tillståndsövergång sker och de händelser som sker som resultat av dessa tillståndsövergångar. Tillstånden i systemet eller objektet som testas är separerade, identifierade och är av ändligt antal. En tillståndstabell visar förhållandet mellan tillstånd, indata och kan synliggöra övergångar som är ogiltiga.

Tester kan skapas för att täcka en typisk sekvens av tillstånd, för att täcka varje tillstånd, för att utföra varje tillståndsövergång, för att utföra specifika sekvenser av tillståndsövergångar som skall testas, eller för att testa ogiltiga tillståndsövergångar.

Tillståndsbaserad testning är ofta använt inom industrin för inbyggd programvara och inom generell teknisk automatisering, men det är en teknik som är användbar för att modellera händelseförlopp i alla typer av system, genom att skapa specifika tillstånd eller testa interaktionen mellan användare och dator (t.ex. för internettillämpningar eller affärsflöden).

4.3.5 Användningsfallsbaserad testning (K2)

Tester kan härledas från användningsfall. Ett användningsfall beskriver interaktionen mellan aktörer, användare eller system, vilket producerar ett resultat av värde för en systemanvändare eller en kund. Användningsfall kan beskrivas på en abstrakt nivå (teknikoberoende affärsflöden på affärsprocessnivå), eller på systemnivå (systemanvändningsfall på funktionalitetsnivå). Varje användningsfall har förvillkor som måste vara uppfyllda för att användningsfallet skall fungera. Varje användningsfall avslutas med ett antal slutvillkor, vilka utgörs av observerbara resultat och systemets sluttillstånd när användningsfallet genomförts. Det krävs också att systemet har ett avsett sluttillstånd när användarscenariot är avslutat. Ett användningsfall har ett huvudflöde, dvs. det mest troliga scenariot, men kan även ha alternativa scenarion.

Användningsfall beskriver “processflödet” genom ett system baserat på dess troliga användning, så testfall som härleds från användningsfall är mycket användbara för att hitta fel i processflödet i den verkliga användningen av systemet. Användningsfall är mycket användbara vid framtagning av acceptanstestfall då kunder eller användare deltar i testfallsframtagningen.De hjälper också till att hitta integrationsfel som orsakas av interaktion mellan och påverkan av, olika komponenter, som man inte hittar då man testar programkomponenterna var för sig. Testdesign utifrån användningsfall kan kombineras med andra specifikationsbaserade tekniker för testdesign.

4.4 *Strukturbaserade eller white-box-tekniker (K4)*	*60 minuter*

Begrepp
Kodtäckning, beslutstäckning, satstäckning, strukturbaserad testning

Bakgrund
Strukturbaserarad testning/white-boxtestning är baserad på att man identifierar strukturen av programmet eller systemet vilket visas genom följande exempel.

- Komponentnivån: Strukturen i en programvarukomponent, dvs. kodsatser, beslut, kodgrenar eller till och med specifika exekveringsvägar.
- Integationsnivån: Strukturen kan vara ett anropsträd, dvs. ett diagram där modulanrop till andra moduler visas
- Systemnivån: Strukturen kan vara en menystruktur, affärsflöde eller en websidas struktur.

I detta avsnitt kommer tre kodrelaterade strukturella testdesigntekniker för kodtäckning, baserat på kodsatser, beslut och kodgrenar att diskuteras. För beslutstestning kommer ett kontrollflödesdiagram att visualisera de olika alternativen för varje villkor.

4.4.1 Kodsatstestning och kodsatstäckning(K4)
I komponenttestning är kodsatstäckning den utvärdering (i procent) som visar hur många exekverbara satser som har exekverats av en svit av testfall. I kodsatstäckning skapar man nya testfall för att exekvera specifika kodsatser, vilket normalt ökar satstäckningen.

Kodsatstäckning är antalet kodsatser som berörs av de (designade eller exekverade) testfallen dividerat med det totala antalet kodsatser i programvarukomponenten.

4.4.2 Beslutstestning och beslutstäckning(K4)
Beslutstäckning, nära relaterat till bågtestning, är utvärderingen (i procent) av antalet beslutsutfall (t.ex. utfallen sant respektive falskt för en IF-sats) som har exekverats av en testfallssvit. Vid beslutstestning skapas testfall som ska exekvera utpekade beslutsutfall. Man kan representera en programvarukod med hjälp av en graf med bågar och noder. En nod med flera utgående bågar som beskriver överföring av kontroll till olika platser i koden representerar en beslutspunkt.

Beslutstäckning är antalet beslutsutfall som berörs av de (designade eller exekverade) testfallen dividerat med det totala antalet beslutsutfall som finns i programvarukomponenten.
Beslutstestning är en form av kontrollflödestestning där ett specifikt kontrollflöde exekveras genom att en specifik exekveringsväg tas genom beslutspunkterma. Beslutstäckning är starkare än kodsatstäckning, eftersom 100% beslutstäckning garanterar 100% kodsatstäckning, men inte tvärtom.

4.4.3 Andra strukturbaserade tekniker (K1)
Utöver beslutstäcking finns andra kodtäckningsmått, t.ex. villkorstäckning och villkorskombinationstäckning.
Kodtäckningskonceptet kan också användas på andra testnivåer, t.ex. på integrationsnivån, där procenten av moduler, komponenter eller klasser som har blivit exekverade av en testfallssvit kan uttryckas som modul-, komponent- eller klasstäckning.

För strukturbaserad testning av kod är verktygsstöd användbart.

4.5 *Erfarenhetsbaserade tekniker (K2)*	*30 minuter*

Begrepp
Utforskande testning, felattack

Bakgrund
Erfarenhetsbaserad testning är när tester skapas baserat på testarens skicklighet och intuition och hans/hennes erfarenhet av liknande applikationer och teknologier. Som komplement till mer systematiska testtekniker, kan dessa tekniker vara mycket värdefulla genom att testfall identifieras, vilka annars inte skulle hittas via de mer formella teknikerna. Dock varierar effektiviteten hos denna typ av testning med erfarenheten hos testaren.

En vanligt förekommande erfarenhetsbaserad teknik är felgissning, i vilken testare förutspår potentiella defekter utifrån sin tidigare erfarenhet. Ett strukturerat sätt att arbeta med felgissning är att göra en lista av möjliga defekter och därefter skriva tester som blottar dessa defekter. Detta systematiska angreppssätt kallas för felattack. Dessa listor på defekter och felsymptom kan byggas upp baserat på erfarenhet, tillgängliga fel eller felsymptombeskrivningar samt från grundläggande kunskap om varför program fallerar.

Utforskande testing är en teknik som bygger på samtidig testdesign, testexekvering, testloggning och inlärning. Den baseras på ett test charter som innehåller testmål och utförs inom väldefinierade tidsgränser. Strategin är mest användbar när det är få eller inte tillräckliga specifikationer och en extrem tidspress, samt som stöd eller komplement till annan, mer formell testning. Tekniken kan tjäna som en kontroll av testprocessen för att försäkra sig om att de mest allvarliga felen hittas.

4.6 *Välja testtekniker (K2)*	*15 minuter*

Begrepp
Inga specifika begrepp

Bakgrund
Val av testtekniker som skall användas beror på ett antal faktorer som, typ av system, lagar, föreskrifter och standarder, kundkrav eller kontrakt, nivå av risk, typ av risk, testdomän, testmål, tillgänglig dokumentation, kunskap hos testarna, tid och budget, utvecklingslivscykel, andvändningsfallsmodeller och tidigare erfarenhet av olika typer av defekter som man hittat. Vissa tekniker är mer användbara vid vissa situationer än andra och vissa tekniker är mest användbara på en testnivå, medan andra kan användas på alla testnivåer.

Under testfallsdesign använder sig testare vanligtvis av en kombination av testtekniker inklusive process-, regelverks- och datadrivna tekniker, för att säkerställa tillräcklig testtäckning av objektet som testas.

Referenser
4.1 Craig, 2002, Hetzel, 1988, IEEE STD 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. Testledning (K3) — 170 minuter

Inlärningsmål för testledning

Efter avslutande av respektive modul nedan skall följande listade målsättningar ha uppnåtts.

5.1. Testorganisation (K2)

IM-5.1.1. Komma ihåg betydelsen av oberoende testning. (K1)
IM-5.1.2. Förklara för- och nackdelar med oberoende testning inom en organisation. (K2)
IM-5.1.3. Komma ihåg hur man väljer personer till testgruppen. (K1)
IM-5.1.4. Komma ihåg typiska arbetsuppgifter för testledare och testare. (K1)

5.2. Testplanering och testuppskattning (K3)

IM-5.2.1. Känna igen olika testnivåer och målsättning med testplanering. (K1)
IM-5.2.2. Sammanfatta syftet och innehållet i dokumenten testplan, testdesignspecifikation och testprocedurspecifikation enligt ‘Standard for Software Test Documentation’ (IEEE 829-1998). (K2)
IM-5.2.3. Klargöra skillnaden mellan konceptuellt skilda angreppssätt för testning såsom analytisk, modellbaserad, systematisk, med process/standard enlighet, dynamisk/heuristisk, konsultativ och ”regression averse”. (K2)
IM-5.2.4. Klargöra skillnaden mellan motiven för testplanering för ett system och planering av testexekveringsordning (K2)
IM-5.2.5. Skriva ett testexekveringsschema för en given mängd av testfall med hänsyn tagen till testfallens prioritet och inbördes logiska förhållanden. (K3)
IM-5.2.6. Kunna lista testförberedelse- och testexekveringsaktiviteter som bör iakttas vid testplannering. (K1)
IM-5.2.7. Komma ihåg typiska faktorer som påverkar arbetsinsatsen relaterad till testning. (K1)
IM-5.2.8. Klargöra skillnaden mellan två begreppsmässigt olika angreppssätt: mätetalsbaserat och expertbaserat. (K2)
IM-5.2.9. Känna igen/motivera fullgoda start- och slutkriterier för specifika testnivåer och grupper av testfall (t.ex. för integrationstest, acceptanstest eller testfall för användbarhetstest). (K2)

5.3. Övervakning och styrning av testförloppet (K2)

IM-5.3.1. Komma ihåg vanliga mätetal som används för övervakning av testförberedelser och exekvering. (K1)
IM-5.3.2. Förklara och jämför mätetal för testrapportering och styrning av test (t.ex. funna och rättade defekter, godkända och inte godkända testfall) relaterat till syfte och användning. (K2)
IM-5.3.3. Summera syftet och innehållet i en slutlig testrapport enligt. ‘Standard for Software Test Documentation’ (IEEE 829-1998). (K2)

5.4. Konfigurationshantering (K2)

IM-5.4.1. Sammanfatta hur konfigurationshantering stödjer testningen. (K2)

5.5. Risker och testning (K2)

IM-5.5.1. Beskriva en risk som ett möjligt problem som hotar uppfyllandet av ett eller flera projektmål för en eller flera intressenter. (K2)
IM-5.5.2. Komma ihåg att en risknivå styrs av sannolikheten att den inträffar och de konsekvenser det medför om den skulle inträffa. (K1)
IM-5.5.3. Skilja mellan projektrisk och produktrisk. (K2)
IM-5.5.4. Visa representativa projektrisker och produktrisker. (K1)
IM-5.5.5. Beskriva, med hjälp av exempel, hur riskanalys och riskhantering kan användas för testplanering. (K2)

5.6. Avvikelsehantering (K3)

IM-5.6.1. Känna igen innehållet i en avvikelserapport enligt ‘Standard for Software Test Documentation’ (IEEE 829-1998). (K1)

IM-5.6.2. Skriva en avvikelserapport för ett fel som upptäckts under testning. (K3)

5.1 *Testorganisation (K2)*	*30 minuter*

Begrepp
Testare, testledare, testchef.

5.1.1 Testorganisation och oberoende (K2)

Att ha oberoende testare i en organisation kan förbättra förmågan att upptäcka fel med hjälp av test och granskningar. Följande är möjliga alternativ för oberoende:

- Inga oberoende testare. Utvecklare testar sin egen kod.
- Oberoende testare i utvecklingsgrupperna.
- Oberoende testgrupp eller testgrupp i en organisation som rapporterar till projektledning eller verkställande ledning.
- Oberoende testare från verksamhetsområdet eller användargrupper.
- Oberoende testspecialister för specifika testtyper, till exempel testare av användbarhet, säkerhet eller certifiering (certifiering av en produkt gentemot standarder och reglementen).
- Oberoende testare som är utlokaliserade internt eller till en extern organisation.

När det gäller stora eller säkerhetskritiska system är det oftast bäst att ha test på flera nivåer, med några eller alla nivåer testade av oberoende testare. Utvecklare kan delta i testningen, speciellt när det gäller lågnivåtester, men deras brist på objektivitet begränsar ofta effektiviteten av detta. Oberoende testare kan ha befogenhet att ställa krav på och definiera testprocesser och regler men testare bör bara ta på sig sådana processrelaterade roller när det finns ett tydligt mandat från ledningen att göra så.

Fördelarna med oberoende testning är bland annat:
- Oberoende testare ser andra och olika defekter, och är förutsättningslös.
- En oberoende testare kan verifiera antaganden andra har gjort under specificering och utveckling av systemet.

Nackdelar:
- Isolering från utvecklingsgruppen (om test är helt oberoende).
- Utvecklare kan känna ett mindre ansvar för kvaliteten.
- Oberoende testare kan ses som en flaskhals eller bli beskyllda för förseningar av en utgåva.

Testarbete kan utföras av personer som har en specifik testroll eller kan göras av någon med en annan roll, till exempel projektledare, kvalitetsansvarig, utvecklare, affärs- eller områdesexpert, infrastruktur eller IT-drift.

5.1.2 Testledarens och testarens arbetsuppgifter (K1)

Denna kursplan täcker två roller, testledare och testare. De aktiviteter och arbetsuppgifter som utförs inom ramen för respektive roll beror på projektets och produktens sammanhang, de individer som innehar rollerna och på organisationen.

Ibland kallas testledaren för testchef eller testkoordinator. Testledarens roll kan utföras av en projektledare, en utvecklingsansvarig, en kvalitetsansvarig eller en chef för en testgrupp. I stora projekt kan det finnas två befattningar: testledare och testansvarig. Testledaren ansvarar vanligtvis för planering, övervakning och styrning av testaktiviteterna som de är definierade i kapitel 1.4.

Typiska uppgifter för en testledare kan vara att:
- Koordinera teststrategi och testplan med projektledare och andra intressenter.
- Skriva eller granska teststrategi för projektet och testpolicy för organisationen.
- Bidra med testsynvinkel i andra projektaktiviteter, till exempel integrationsplanering.

- Planera testningen – med hänsyn tagen till sammanhanget och förstå målen och riskerna – inklusive val av angreppssätt för test, uppskattning av tid, arbetsmängd och kostnad för test, få fram resurser, definition av testnivåer, cykler, angreppssätt, målsättning och planering av avvikelserapporteringsprocessen.
- Initiera specificering, förberedelse, realisering och genomförande av tester, övervaka testresultat och kontrollera avslutskriterier.
- Anpassa planeringen beroende på testresultat och förlopp (ibland dokumenterat i status-rapporter) och vidta nödvändiga åtgärder för att kompensera för uppkomna problem.
- Sätta upp lämplig konfigurationshantering av testprodukter för spårbarhet.
- Införa passande mätetal för att mäta testförloppet i och för att utvärdera kvaliteten på testningen och produkten.
- Besluta vad som skall automatiseras, till vilken grad och hur.
- Välja verktyg för teststöd och organisera utbildning av testare för användningen av verktygen.
- Besluta hur testmiljön skall realiseras.
- Skriva slutlig testrapport baserat på resultat och information från testningen.

Typiska uppgifter för en testare kan vara:

- Granska och bidra till utveckling av testplaner.
- Analysera, granska och utvärdera användarkrav, specifikationer och modeller ut testbarhets-synpunkt.
- Skapa testspecifikationer.
- Sätta upp testmiljöer (ofta i samarbete med systemadministratörer och nätverksansvariga).
- Förbereda och ta fram testdata.
- Skapa tester på alla nivåer, genomföra och logga tester, utvärdera testresultat och dokumentera avvikelser från förväntat resultat.
- Använda testverktyg för administration, styrning och övervakning enligt direktiv.
- Automatisera tester (kan kräva stöd av utvecklare eller automatiseringsexpert).
- Mäta prestanda på komponenter eller system (om tillämpligt).
- Granska testfall utvecklade av andra.

Personer som arbetar med testanalys, testdesign, specifika testtyper eller automatisering av test kan vara specialister i dessa roller. Beroende på testnivån och vilka risker som är relaterade till produkten och projektet kan andra personer ta rollen som testare och på så sätt etablera en viss nivå av oberoende. Till exempel kan testare på komponent- och integrationsnivå utgöras av utvecklare medan testare på acceptanstestnivå utgöras av områdesexperter och användare och testare av acceptanstest för drift kan vara driftpersonal.

5.2 *Planering och testuppskattning (K3)*	*40 minuter*

Begrepp
Angreppssätt för testning, teststrategi.

5.2.1 Testplanering (K2)

Detta kapitel omfattar syftet med testplanering i utvecklings- och införandeprojekt och för förvaltningsaktiviteter. Planeringen kan dokumenteras i en övergripande testplan (master test plan) och i separata testplaner för varje nivå, till exempel systemtestplan och acceptanstestplan. Huvuddragen i ett testplaneringsdokument finns beskrivet i ”Standard for Software Test Documentation” (IEEE STD 829-1998).

Planeringen påverkas av organisationens testpolicy, testningens omfattning, målsättning, risker, begränsningar, hur kritisk produkten är, testbarhet och tillgång till resurser. Ju mer projektet och testplaneringen framskrider, desto mer information finns tillgänglig och mer detaljer kan läggas till i planen.

Testplanering är en kontinuerlig aktivitet och görs under alla livscykelprocesser och alla aktiviteter. Återkoppling från testaktiviteterna används för att se ändringar i risker så att planeringen kan justeras.

5.2.2 Aktiviteter i testplaneringen (K3)

Testplaneringsaktiviteter för ett komplett system eller delar av ett system kan omfatta:

- Avgränsa, bestämma risker och identifiera målet med testningen.
- Definieria övergripande tillvägagångssätt för testningen inklusive definition av testnivåer, start- och avslutningskriterier.
- Integrera och koordinera testaktiviteterna med livscykelaktiviteterna för programvaruutveckling: anskaffning, leverans, utveckling, drift och underhåll.
- Besluta om vad som skall testas, vilka roller som skall utföra testaktiviteterna, hur testaktiviteterna skall utföras, hur resultatet skall utvärderas.
- Fastställa tidpunkt för aktiviteterna testanalys och testdesign.
- Fastställa tidpunkt för testrealisering, exekvering och utvärdering.
- Tilldela resurser för de olika definierade arbetsuppgifterna.
- Definiera mängd, detaljnivå, struktur och mallar för testdokumentationen.
- Välja mätetal för övervakning och styrning av testförberedelser, testexekvering, felrättning och risker.
- Besluta om hur detaljerade testprocedurerna skall vara för att tillhandahålla tillräcklig information som stöd till reproducerbara testförberedelser och exekvering.

5.2.3 Startkriterier

Startkriterier definierar när testningen kan starta som t.ex. i början av en testnivå eller när en mängd tester är klara för körning.

Typiska startkriterier kan vara:

- Testmiljöns tillgänglighet och hur komplett den är
- Testverktygens grad av färdigställande i testmiljön
- Tillgänglighet till testbar kod
- Tillgänglighet till testdata

5.2.4 Avslutskriterier (K2)

Avslutskriterier är i förväg bestämda nivåer hos ett antal mätetal som jämförs med de mätningar som testningen resulterar i för att definierar när testningen skall avslutas, till exempel vid slutet av en testnivå eller när en uppsättning av testfall har uppnått ett speciellt mål.

Typiska avslutskriterier kan vara:
- Testkvalitetsrelaterade mätetal, som kodtäckning, täckning av funktionalitet eller risker.
- Uppskattningar av systemets feldensitet och/eller tillförlitlighet.
- Kostnad.
- Kvarvarande risker, t.ex. inte åtgärdade fel eller otillräcklig testtäckning inom vissa områden.
- Tidsplaner, t.ex. sådana som baseras på marknadskrav.

5.2.5 Testuppskattning (K2)

Två tillvägagångssätt för uppskattning av mängden testarbete är:
- Mätetalsbaserat angreppssätt: Uppskattning som bygger på mätresultat från tidigare eller liknande projekt eller baserat på vanligt förekommande värden.
- Erfarenhetsbaserat angreppssätt: Uppskattning av uppgiften baseras på uppskattning gjord av uppgiftens ägare eller av en expert.

När testarbetsmängden väl är beräknad, kan resurser identifieras och en tidsplan utarbetas.

Testinsatsen kan bero på ett flertal faktorer, såsom:
- Produktens egenskaper:
  kvaliteten på specifikationer och annan information som används för testmodeller (t. ex. testbas), produktens storlek, testdomänens komplexitet, kraven på tillförlitlighet, säkerhet och dokumentation.
- Utvecklingsprocessens egenskaper:
  stabiliteten i organisationen, använda verktyg, testprocess, färdigheten hos inblandande personer och tidspressen.
- Testutfallet:
  antalet defekter och mängden omarbetningar som krävs.

5.2.6 Teststrategi, testangreppssätt (K2)

Implementeringen av teststrategin för ett visst projekt kallas testangreppssätt. Det definieras och förfinas i testplanen och testdesignen. Vanligtvis baseras testangreppssättet på beslut tagna kring (test)-projektets mål och identifierade risker. Testangreppssättet utgör startpunkten för planering av testprocessen, val av testdesigntekniker och testyper som ska användas och även definition av start- och slutkriterier.

Det valda angreppssättet beror på sammanhanget, och hänsyn tas till risker, faror och säkerhet, tillgängliga resurser och kompetens, teknologier, typ av system (t.ex. specialanpassat eller COTS), testmålsättning, regler och föreskrifter.

Typiska angreppssätt inkluderar:
- Analytiskt angreppssätt, såsom riskbaserad testning där testningen styrs mot områden med största risker.
- Modellbaserat angreppssätt, såsom stokastisk testning där statistisk information om felfrekvens används (t.ex. reliability growth models) eller användning (såsom användarprofiler).
- Metodiskt angreppssätt som exempelvis är avvikelsebaserat (såsom felgissning och felattacker), baserat på erfarenhetsbaserade checklistor eller baserat på kvalitetsegenskaper.
- Process- eller standardanpassat angreppssätt, t.ex. specificerade av industristandard eller olika lättviktsmetoder (agila).
- Dynamiskt eller heuristiskt angreppssätt, som utforskande testning där testningen är mer reaktiv till händelser än planerade i förväg och där exekveringen och utvärderingen utförs parallellt.
- Konsultativt angreppssätt, exempelvis då testtäckningen primärt drivs med råd och vägledning från tekniska experter och/eller affärsområdesexperter utanför testgruppen.

- Regressionsvänligt sätt, där existerande testunderlag återanvänds, omfattande automatisering av funktionella regressionstester och standardiserade testsviter.

Olika angreppssätt kan kombineras, till exempel ett riskbaserat dynamiskt arbetssätt.

5.3 *Övervakning och styrning av testförloppet (K2)*	*20 minuter*

Begrepp
Feltäthet, felfrekvens, teststyrning, testövervakning, slutgiltig testrapport.

5.3.1 Övervakning av testprocessen (K1)

Syftet med övervakning av testprocessen är att ge återkoppling från testaktiviteterna och göra dessa synliga. Informationen kan samlas in manuellt eller med hjälp av verktyg och kan användas för att mäta avslutskriterier som exempelvis testtäckning. Mätetal kan också användas till att bedöma testförloppet i förhållande till tidsplan och budget.
Vanligt förekommande mätetal:

- Upparbetad procentuell arbetsmängd för att skriva testfall (eller procentuellt antal förberedda testfall).
- Upparbetad procentuell arbetsmängd i förberedelser för testmiljön.
- Genomförande av testfall (t.ex. antal körda/ej körda testfall och godkända/ej godkända testfall).
- Information om fel (t.ex. feltäthet, fel funna och åtgärdade, felfrekvens och resultat från omtester).
- Testtäckning av krav, risker och kod.
- Testarnas subjektiva förtroende till produkten.
- Datum för testmilstolpar.
- Kostnader för test, inkluderande kostnaderna jämfört med vinsten av att hitta nästa fel eller att köra nästa test.

5.3.2 Testrapportering (K2)

Testrapportering innebär att sammanfatta information om testinsatserna, t.ex.:

- Rapportering av vad som har hänt under testperioden, t.ex. datum när avslutskriterier uppnåddes.
- Analysinformation baserad på mätetal som underlag för kommande beslut. Exempel på sådan rapportering är antal kvarvarande fel, ekonomiska fördelar med att fortsätta testningen, olösta risker och graden av tilltro på den testade programvaran.

Dispositionen hos en slutlig testrapport finns i ”Standard for Software Test Documentation” (IEEE 829-1998).

Mätetal skall samlas in under och i slutet av, en testnivå för att kunna utvärdera:

- Till vilken grad testmålen uppnåtts för den aktuella testnivån.Lämpligheten hos det använda testangreppssätten.
- Verkningsgraden av testningen med avseende på de definierade testmålen

5.3.3 Teststyrning (K2)

Teststyrning innefattar de åtgärder som genomförs för att guida eller korrigera ett projekt som ett resultat av insamlad information eller genomförda mätningar. Åtgärderna kan avse alla testaktiviteter och kan påverka andra aktiviteter eller arbetsuppgifter inom programvarans livscykel.

Exempel på aktiviteter inom teststyrning:

- Fatta beslut baserat på information från testövervakning.
- Omprioritera tester när en identifierad risk uppträder (till exempel försenad leverans av programvaran).
- Ändra tidsplanen för test beroende på tillgängligheten av testmiljön.
- Sätta ett startkriterium att felrättningar skall ha omtestats (omtest av felrättning) av en utvecklare innan de accepteras i ett bygge.

5.4 *Konfigurationshantering (K2)*	*10 minuter*

Begrepp
Konfigurationshantering, versionshantering.

Bakgrund
Syftet med konfigurationshantering är att skapa och underhålla integriteten hos produkterna (komponenter, data och dokumentation) i programvaran eller systemet genom projektets och produktens livscykel.

När det gäller testning så kan konfigurationshanteringen säkra att:

- Alla beståndsdelar i testvaran är identifierade, versionshanterade, spårbara för ändringar, kopplade till varandra och kopplade till utvecklingsprodukter (testobjekt) så att spårbarhet kan underhållas genom hela testprocessen.
- Alla dokument och programvaruprodukter refereras otvetydigt till i testdokumentation.

Konfigurationshanteringen hjälper testaren att unikt identifiera (och att reproducera) den testade produkten, testdokument, testerna och testexekveringsplattform.

Under testplaneringen skall konfigurationshanteringsrutiner och infrastruktur (verktyg) väljas, dokumenteras och införas.

5.5 *Risk och testning (K2)*	*30 minuter*

Begrepp
Produktrisk, projektrisk, risk, riskbaserad testning.

Bakgrund
Risk kan definieras som sannolikheten att en händelse, hot eller situation inträffar, med dess oönskade konsekvenser. Nivån på risken bestäms av sannolikheten att den oönskade händelsen inträffar och dess konsekvens (skadan som uppstår på grund av händelsen).

5.5.1 Projektrisker (K2)
Projektrisker är sådana risker som omfattar projektets förmåga att leverera enligt dess målsättningar, som:

- Organisatoriska problemställningar:
  - Brist på utbildning, kompetens och personal
  - personliga problem
  - politiska problem, som
    - problem med testarnas kommunikation om deras behov och testresultat;
    - underlåtenhet att följa upp information som hittats i samband med test och granskning (som t ex kan leda till att arbetsmetoder för utveckling och test inte förbättras).
  - Felaktiga attityder eller förväntningar på testningen (t.ex. att inte uppskatta värdet av att hitta fel under test).
- Tekniska problem:
  - Svårighet att definiera rätt krav.
  - Tekniska begränsningar som medför att kraven inte kan uppfyllas till fullo.Testmiljön inte färdig i tid.
  - Försenade datakonvertingar, datamigrerings planering och utveckling och testning av datakonverterings- och datamigreringsverktyg.
  - Kvaliteten på konstruktionen, koden, konfigurationsdata, testdata och testerna.
- Leverantörsproblem:
  - Fel funna i tredjepartsprodukter;
  - kontraktsproblem.

För att analysera, hantera och mildra dessa risker följer testchefen/testledaren väl etablerade projektledningsprinciper. Standarden ”Standard for Software Test Documentation”, IEEE STD 829-1998, kräver att risker och åtgärder anges i testplanen.

5.5.2 Produktrisker (K2)
Potentiella felområden (oönskade framtida händelser eller risker/faror) i programvara eller system benämns produktrisker, eftersom de utgör risker för produktkvaliteten, t.ex.:

- Många fel i den levererade programvaran.
- Möjligheten att programvaran/hårdvaran kan skada individer eller företag.
- Programvaruegenskaper som inte motsvarar kraven (t.ex. funktionalitet, tillförlitlighet, användbarhet och prestanda).
- Dålig dataintegritet och kvalitet (t.ex. orsakade av felaktigt genomförd datamigrering, datakonvertering eller datatransport, eller brott mot datastandarder).
- Programvara som inte utför det den är avsedd för.

Risker används för att besluta var testning skall starta och var ytterligare testning behöver göras. Tester görs för att minska risken för eller minimera konsekvenserna av en oönskad händelse.

Produktrisker är en speciell typ av risk som påverkar om projektet kommer att lyckas. Genom att använda testning som en riskhanteringsaktivitiet får man information om kvarvarande risker genom att mäta organisationens förmåga att åtgärda kritiska defekter och effektiviteten hos åtgärdsplanen.
Ett riskbaserat angreppssätt vid testningen ger proaktiva möjligheter att minska nivån på produktriskerna, redan vid de första stegen i ett projekt. Det omfattar identifieringen av produktriskerna och att använda riskerna som ett styrmedel vid testplanering, projektstyrning, testspecificering, testförberedelser och testexekvering. I ett riskbaserat angreppssätt används de identifierade riskerna till:

- Att bestämma vilka testekniker som skall användas.
- Att avgöra hur omfattande tester som behövs.
- Att prioritera testaktiviteterna så att kritiska defekter hittas så tidigt som möjligt.
- Att avgöra om andra aktiviteter än testningen kan användas för att minska riskerna (t.ex. tillhandahålla utbildning för oerfarna utvecklare).

Riskbaserad testning baseras på den kollektiva kunskapen och inblicken hos projektets intressenter för att identifiera de risker och testnivåer som behövs för att adressera dessa risker.
För att försäkra sig om att sannolikheten för produktfel är minimerad, tillhandahåller riskhanteringen ett ordnat arbetssätt för att:

- Utvärdera (och regelbundet omutvärdera) vad som kan gå fel (risker).
- Bestämma vilka risker som är viktiga att hanteras.
- Införa åtgärder för att hantera dessa risker.

Dessutom kan testning stödja identifieringen av nya risker, hjälpa till med vilka risker som skall minskas och minska osäkerheten om risker.

5.6 *Avvikelsehantering (K3)*	*40 minuter*

Begrepp
Avvikelseloggning, avvikelsehantering, avvikelserapport

Bakgrund
Eftersom en av målsättningarna med testning är att hitta defekter, måste skillnaden mellan aktuella och förväntade resultat loggas i form av avvikelser. Avvikelser måste utredas då de kan bero på defekter. Lämpliga åtgärder för att avlägsna avvikelser och defekter skall definieras. Avvikelser och defekter skall spåras från upptäckt och klassificering till åtgärd och bekräftelse av lösningen. För att kunna hantera alla avvikelser till avslut, måste organisationen upprätta en avvikelsehanteringsprocess inklusive regler för klassificering.

Avvikelserapporter kan skapas under utveckling, granskning, testning eller användandet av programvaruprodukten. De kan gälla problem i koden, i ett system som körs eller i alla typer av dokumentation, inkluderande kravspecifikation, utvecklingsdokument, testdokument och användardokument såsom hjälpsystemet eller installationsanvisningar.

Avvikelserapporter har följande målsättningar:
- Ge utvecklare och andra återkoppling om problem för att på så sätt möjliggöra identifiering, isolering och den korrigering som är nödvändig.
- Ge testledare ett hjälpmedel att med spårbarhet följa det testade systemets kvalitet och övervaka hur testarbetet fortskrider.
- Ge idéer för förbättring av testprocessen.

Detaljer i avvikelserapporten kan vara:
- Datum för utfärdande, utfärdande organisation och författare.
- Förväntat och aktuellt resultat.
- Identifiering av ett testelement (konfigurationselement) och miljö.
- I vilken del av livscykelprocessen för programvara eller system felet upptäcktes.
- Beskrivning av avvikelsen för att möjliggöra reproducering och en lösning, inkluderar loggar, databasdump eller skärmdump.
- Omfattning och allvarlighetsgrad ur intressenternas perspektiv.
- Allvarligheten med avseende på den påverkan felet har på systemet.
- Angelägenhet/prioritet för att åtgärda felet.
- Status på avvikelserapporten (t.ex. öppen, vilande, dubblett, väntar på att åtgärdas, åtgärdad och väntar på omtest, eller stängd).
- Slutsatser, rekommendationer och godkännande.
- Globala faktorer, t.ex. andra områden som kan påverkas av felrättningen.
- Ändringshistorik, t.ex. åtgärdsstegen gjorda av projektmedlemmar för att isolera, åtgärda och bekräfta att felet är åtgärdat.
- Referenser, som inkluderar identiteten av den testfallspecifikation som påvisade problemet.

Strukturen hos en avvikelserapport täcks också av ”Standard for Software Test Documentation” (IEEE 829-1998).

Referenser

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. Verktygsstöd vid test (K2)	80 minuter

Inlärningsmål för testning med verktygsstöd.
Följande målsättningar beskriver vad som förväntas kunnas efter varje modul.

6.1. Olika typer av testverktyg (K2)

IM-6.1.1. Klassificera olika typer av testverktyg baserat på deras ändamål och på aktiviteter i testprocessen och i livscykeln för en programvara. (K2)
IM-6.1.3. Förklara termen testverktyg och ändamålet med verktygsstöd

6.2. Effektivt användande av testverktyg: möjligheter och risker (K2)

IM-6.2.1. Summera potentiella risker och möjligheter med att automatisera test, samt verktygsstöd för testningen.
IM-6.2.2. Komma ihåg speciella faktorer gällande testexekveringsverktyg, statisk analys och testledningsverktyg.

6.3. Införande av testverktyg i en organisation (K1)

IM-6.3.1. Beskriva huvudprinciperna för införande av testverktyg i en organisation.
IM-6.3.2. Beskriva målen för en "proof-of-concept" vid verktygsutvärdering respektive en förstudie vid införande av verktyg. (K1)

Förstå faktorer, utöver själva inköpet, som har betydelse för att ett verktygsstöd ska lyckas.

6.1 *Olika typer av testverktyg (K2)*	*45 minuter*

Begrepp
Konfigurationshanteringsverktyg, dvs. CM-verktyg, verktyg för testtäckning, debuggverktyg, drivrutiner, verktyg för dynamiskanalys, verktyg för felhantering, verktyg för lasttester, verktyg för modellering, verktyg för övervakning, verktyg för prestandatester, probeffekt, kravhanteringsverktyg, verktyg för granskningsstöd, säkerhetsverktyg, verktyg för statiskt analys, verktyg för stresstester, stubbe, testjämförelse, testdataverktyg, verktyg för testdesign, testmiljö, verktyg för testexekvering, verktyg för testhantering, enhetstestverktyg.

6.1.1 Verktygsstöd inom testning (K2)

Testverktyg kan användas i en eller flera aktiviteter som stödjer testning, och innefattar:
1. Verktyg som används explicit för testning, såsom testexekveringsverktyg, verktyg för generering av testdata och verktyg för resultatjämförelse
2. Verktyg för stöd i testprocessen såsom verktyg för hantering av testfall, testresultat, data, krav, avvikelser, etc., även rapportering och övervakning av testexekvering
3. Verktyg som används för insamling av information genom observation, eller med ett enklare ord, ***utforskning***, Det kan till exempel vara verktyg för övervakning av en applikations filaktiviteter
4. Övriga verktyg som kan användas för att stödja testning (kalkylark räknas också in som verktyg i detta sammanhang)

Verktygsstöd för testning kan ha ett eller flera av följande syften beroende på sammanhanget:
- Förbättra effektiviteten i testaktiviteter genom att automatisera repetitiva uppgifter och stödja manuella uppgifter såsom testplanering, testdesign, testrapportering och övervakning
- Automatisera aktiviteter som kräver stora resurser när dessa utförs manuellt (t.ex., statisk testning)
- Automatisera aktiviteter som inte kan utföras manuellt (t.ex., storskaliga prestandatester av klient-server applikationer)
- Förbättra tillförlitligheten i testningen (t.ex., genom att automatisera jämförelser av stora datamängder eller simulering av beteende)

Begreppet testramverk används inom testbranschen med åtminstone tre olika betydelser:
- Återanvändbara och utökningsbara testbibliotek som kan användas till att bygga testverktyg (också kallad testexekveringsplattform)
- Vissa typer av stöd för testautomatisering t.ex., datadriven eller nyckelordsdriven
- Generellt stöd vid testfallsexekvering

I denna kursplan används begreppet "testramverk" i de två första betydelserna och beskrivs i Kapitel 6.1.6 (återanvändbara bibliotek) respektive Kapitel 6.2.2 (design av automation).

6.1.2 Klassificering av testverktyg (K2)

Det finns ett antal olika typer av testverktyg som stödjer olika aspekter av test. Verktyg kan klassificeras enligt flera kriterier såsom syfte, kommersiell/Free/Open-Source/Shareware, använd teknologi, etc. Testverktygen i denna kursplan är klassificerade enligt de testaktiviteter som de stödjer.

För vissa testverktyg är det uppenbart att de stödjer endast en aktivitet medan andra kan stödja flera aktiviteter fast de är klassificerade under den aktivitet som verktyget är starkast associerat till. Verktyg från en tillverkare, speciellt om de är utvecklade för att fungera tillsammans, kan vara samlade i en verktygssvit.

Vissa typer av testverktyg kan vara inkräktande, vilket betyder att de kan påverka det faktiska testresultatet. Till exempel kan val av verktyg ge en påverkan vid tidsmätningar genom de extrainstruktioner som verktyget måste exekvera eller vid kodtäckningsmätningar då olika verktyg

mäter på olika sätt. Konsekvensen när ett inkräktande testverktyg påverkar testresultatet kallas för probeffekt.

Några verktyg erbjuder stöd som är mer lämpligt för utvecklare (t.ex. under komponent- eller komponentintegrationstest). Dessa verktyg är markerade med ”(U)” i klassificeringen nedan.

6.1.3 Verktygsstöd för hantering av testning och tester (K1)

Administrativa verktyg används för alla testaktiviteter under programutvecklingens hela livscykel.

Verktyg för testhantering
Dessa verktyg tillhandahåller gränssnitt mot testexekverings-, felspårnings- och kravhanteringsverktyg, och stödjer kvantitativ analys och rapportering av testobjekten. De stödjer även spårbarhet av testobjekt gentemot kravspecifikationen och kan innehålla stöd för versionshantering direkt eller indirekt via gränssnitt mot en extern versionshantering.

Kravhanteringsverktyg
Dessa verktyg lagrar krav, stödjer prioritering av kraven och tillåter spårbarhet mellan krav och individuella testfall. Verktygen kan även ibland detektera inkonsistenser mellan olika krav eller krav som saknas.

Verktyg för hantering av avvikelser (felhanteringsverktyg)
Dessa verktyg lagrar och hanterar avvikelserapporter, vilka kan innehålla beskrivningar av felsymptom, ändringsbegäran, uppfattade problem och anomalier. Verktygen hjälper även till med att hantera livscykeln för avvikelser, och kan i vissa fall även stödja statistisk analys.

Verktyg för konfigurationshantering
Även om dessa verktyg inte direkt är ett testverktyg är de nödvändiga för att hålla ordning på olika versioner och byggen av programvaror och tester, speciellt om man har flera konfigurationer av testobjektet eller multipla testmiljöer beroende på olika versioner av operativsystem, kompilatorer eller editorer.

6.1.4 Verktyg för statisk testning (K1)

Verktyg för statisk testning är ett kostnadseffektivt hjälpmedel för att hitta defekter i de tidiga stegen i utvecklingsprocessen.

Verktyg för stöd av granskningsprocessen
Dessa verktyg kan lagra information om granskningsprocessen, lagra och kommunicera granskningsresultat, rapportera fel och arbetsinsats, hantera referenser till granskningsregler och/eller checklistor, samt hålla ordning på spårbarheten mellan dokument och källkod. De kan också vara en hjälp vid datorstödda granskningar vilket är värdefullt när de som arbetar med detta är geografiskt spridda.

Verktyg för statiska analys (U)
Dessa verktyg stödjer utvecklare, testare och kvalitetssäkringspersonal i att hitta fel innan dynamisk testning startar. Verktygsstöd säkerställer att organisationen följer kodningsstandarder (inklusive fokus språkkonstruktioner för att öka datasäkerheter) samt innefattar analys av strukturer och beroenden. De kan även vara till hjälp för planering eller riskanalys genom att mäta olika kodegenskaper (t.ex. komplexitet).

Verktyg för modellering (U)
Dessa verktyg används för att validera modeller av programvara (t.ex. fysiska datamodeller av en relationsdatabas), genom att identifiera inkonsekvens och hitta defekter. Verktygen kan också i vissa fall användas för att generera modellbaserade testfall.

6.1.5 Verktyg för stöd av testspecificering (K1)

Verktyg för testdesign
Dessa verktyg genererar indata till, eller exekverbara testfall och/eller test orakel från krav, grafiska användargränssnitt, utvecklingsmodeller (tillstånd, data eller objekt) eller från koden.

Verktyg som förbereder testdata
Dessa verktyg kan manipulera innehållet i databaser, filer eller dataöverföringar för att skapa testdata som skall användas under testexekveringen. En fördel med dessa verktyg är att de kan garantera att riktiga data som har överförts till testmiljön har avpersonifierats av integritetsskäl.

6.1.6 Verktyg för stöd av testexekvering och loggning (K1)

Verktyg för testexekvering
Dessa verktyg ger möjlighet att exekvera testfall hel- eller halvautomatiskt. Verktygen exekverar skript, vilka hämtar förberedda indata och förväntade resultat och oftast producerar en testlogg vid varje testkörning.De kan också användas för att spela in testfall, och stödjer ofta skriptspråk eller GUI-baserade konfigurationer för parametrisering av data och andra specialiseringar av testfall.

Testexekveringsplattform/ramverktyg för komponenttest (U)
En testexekveringsplattform för enhetstest eller ramverk underlättar test av komponenter eller del av ett system genom att den simulerar den miljön där testobjektet exekveras och genom attden tillhandahåller stubbar eller drivrutiner.

Testjämförare/testkomparator
Dessa verktyg kan identifiera skillnader mellan olika filer, databaser eller testresultat. Testexekveringsvektyg innehåller normalt dynamiska jämförare, men jämförelse efter exekvering kan göras av ett separat jämförelseverktyg. En testjämförare kan använda ett testorakel, speciellt om det är frågan om automatisering

Verktyg för kodtäckning (U)
Dessa verktyg, inkräktande eller icke-inkräktande, mäter hur stor del i procent av kodstrukturen (t.ex. kodsatser, kodgrenar eller kodvillkor samt komponent eller funktionsanrop, som täckts av testningen.

Verktyg för informationssäkerhet
Dessa verktyg utvärderar säkerhetsegenskaperna hos programvara, t.ex. förmågan att skydda konfidentiellt data, integritet, identifiering, behörighet och tillgänglighet. Verktyg för informationssäkerhet är ofta anpassade till specifika teknologier, plattformar och syften.

6.1.7 Verktygsstöd för prestanda och övervakning (K1)

Verktyg för dynamiska analys (U)
Verktyg för dynamisk analys hittar fel som enbart uppträder när programvaran exekveras, såsom tidsproblem eller minnesläckor. Verktygen används typiskt under komponenttest eller komponentintegration och vid test av middleware.

Verktyg för prestandatest/lasttest/stresstest
Verktyg för prestandatest övervakar och rapporterar hur ett system beter sig under olika simulerade driftssituationer med avseende på antalet samtidiga användare, deras användarmönster, frekvens och relativt utnyttjande i procent av transaktioner. Simulering av last åstadkoms genom att skapa virtuella användare som, utspridda över olika typer av testmaskiner (lastgeneratorer), utför utvalda transaktioner.
Verktyg för övervakning
Verktyg för övervakning analyserar, verifierar och rapporterar kontinuerligt användandet av specifika systemresurser och varnar för vissa typer av driftsproblem.

6.1.8 Verktygsstöd för specifika applikationsområden (K1)

Utvärdering av datakvalitet

Data är en viktig del i vissa projekt såsom datakonverterings- och datamigreringsprojekt samt i tillämpningar såsom datamagasinering. Dess egenskaper kan variera med avseende på kritiskhet och volym. I sådana situationer måste verktyg användas för att utärdera datakvaliteten genom att granska och verifiera reglerna för konvertering och migrering så att slutgiltiga data är korrekta, fullständiga och överensstämmer med fördefinierade standarder som är tillämpbara. Speciella verktyg finns även för användbarhetstestning.

6.2 *Effektivt användande av verktyg: potentiella fördelar och risker (K2)*	*20 minuter*

Begrepp
Datadriven testning, nyckelordsdriven testning, testskriptspråk

6.2.1 Möjliga fördelar och risker med att använda verktygsstöd i testningen (K2)

Att enbart köpa in eller hyra ett verktyg garanterar inte att man kommer att lyckas med det. Varje typ av verktyg kan kräva extra insatser för att få verkliga och bestående fördelar. Det finns möjliga fördelar med att använda verktyg i testningen men det finns också risker.

Möjliga fördelar med att använda verktyg inkluderar:

- Minskning av repetitivt arbete (t.ex. att utföra regressionstester, att återinmata testdata och att kontrollera att kodningsstandarder efterlevs).
- Bättre jämnhet i den resulterande testkvaliteten och ökad möjlighet till återupprepning (t.ex. testfall exekverade i samma ordning, med samma frekvens, vid samma tidpunkt eller tester härledda från krav.).
- Ökad objektivitet i bedömningen av data och testresultat (t.ex. statiska mätningar och täckning).
- Lättare att få tillgång till information om tester och testning (t.ex. statistik och grafer gällande testprogress, felfrekvens och prestanda).

Risker med att använda verktyg inkluderar:

- Orealistiska förväntningar på verktyget (med avseende på funktionalitet och hur lätt det är att använda).
- Underskattning av kostnad och arbetsinsats vid introduktion av ett verktyg (inkusive utbildning och externa experter).
- Underskattning av den tid och insats som behövs för att nå betydande och bestående fördelar med hjälp av verktyget (inklusive behovet av ändringar i testprocessen) och kontinuerliga förbättringar av det sätt som verktyget används.
- Underskattning av den insats som krävs för att underhålla de tillgångar som genererats av verktyget.
- Övertro på verktyget (att verktyget kan ersätta testdesign eller manuella tester, där dessa faktiskt fungerar bättre).
- Att versionshantering försummas av de testtillämpningar som läggs in i verktyget.
- Att beroende och problem i interaktionen mellan kritiska verktyg försummas. Exempel: kravhanteringsverktyg, versionshanteringsverktyg och verktyg från flera olika leverantörer.
- Att verktygsleverantörer försvinner, slutar med verktyget eller säljer verktyget till annan leverantör.
- Dåligt gensvar från tillverkare vid support, uppgradering och korrigering av fel.
- isk för att verktygsprojekt som bedrivs som Open-Source/Free projekt läggs ner. Oförutsedda risker, som t.ex. frånvaron av stöd för en ny plattform

6.2.2 Särskilda hänsynstaganden för vissa typer av verktyg (K1)

Verktyg för testexekvering
Testexekveringsverktyg styr exekveringen av testobjekten via automatiserade testskript. Den här typen av verktyg kräver oftast betydande insatser för att uppnå meningsfulla mål.

Att skapa testfall genom att spela in manuell testning kan vara attraktivt, men denna ansats kan inte skalas upp till ett stort antal automatiserade testskript. Ett inspelat skript är en linjär representation med specifika data och åtgärder som är hårdkodade i varje skript. Den här typen av skript kan bli instabil när oväntade händelser inträffar.

Vid datadriven testning separeras indata ut, vanligtvis till ett kalkylprogram, och använder ett mer generiskt testskript som kan läsa indata och använda samma testskript med olika data. Testare som inte är bekanta med skriptspråk kan fortfarande skapa testdata till dessa fördefinierade skript.

Det finns även andra tekniker som kan användas vid datadriven testning. I stället för att använda kalkylblad med fast bestämda kombinationer av data så genereras indata till applikationen med hjälp av en algoritm som baseras på konfigurerbara parametrar vid körning. Ett exempel är en algoritm som genenrerar slumpvisa användaridentiteter baserat på ett känt slumptalsfrö för att göra det möjligt att återskapa en viss uppsättning data.

Vid ett nyckelordsdrivet angreppssättinnehåller kalkylprogrammet, förutom testdata, nyckelord som beskriver vilka åtgärder som skall göras (kallas också skriptkommandon), Testare kan då (även om de inte är bekanta med skriptspråk) definiera tester genom att använda nyckelorden. Nyckelorden kan skräddarsys till den applikation som skall testas.

Teknisk expertis på skriptspråk behövs för alla angreppssätt (antingen testare eller specialister på testautomatisering).

Det är alltid nödvändigt att lagra förväntat testresultat för senare jämförelse med verkligt, oavsett vilken skriptteknik som används.

Verktyg för statisk analys
Verktyg för statisk analys kan användas för att säkerställa att en kodningsstandard efterlevts. Användning av dessa verktyg på befintlig kod kan resultera i en stor mängd varningar. Dessa varningar hindrar inte att koden kompileras till exekverbara program, men idealiskt bör de tas om hand, så att koden i framtiden blir enklare att underhålla. I ett läge med en stor mängd varningar kan en effektiv ansats vara att införa verktyget stegvis genom att inledningsvis ignorera vissa typer av varningar genom att filtrera bort dessa och därefter gradvis ta bort filtren.
Verktyg för testhantering
Verktyg för testhantering måste kunna samverka med andra verktyg eller kalkylprogram för att producera information på ett format som passar organisationens behov.

6.3 *Införande av ett verktyg i en organisation (K1)*	*15 minuter*

Begrepp
Inga speciella begrepp.

Bakgrund
De viktigaste frågeställningarna när ett verktyg väljs för en organisation omfattar:

- Utvärdering av organisationens mognad, styrka och svagheter samt identifiering av möjligheterna till en förbättring av testprocessen med hjälp av verktyg.
- Formulering av tydliga och objektiva krav för utvärdering av verktyget.
- Genomförande av ett "proof-of-concept" genom att använda testverktyget redan under utvärderingen för att säkerställa att det verkligen fungerar i faktiska testmiljön med befintlig infrastruktur och för att identifiera eventuella förändringar av testmiljön och infrastrukturen för att verktyget ska generera så stora fördelar som möjligt.
- Utvärdering av verktygsleverantören (även med avseende på utbildning, stöd och kommersiella aspekter) eller leverantörer av supporttjänster när det gäller icke-kommersiella verktyg.
- Identifiering av interna krav på handledning och rådgivning för användning av verktyget.
- Utvärdering av utbildningsbehov baserat på den nuvarande kompetensen inom testautomatisering för testgruppen.
- Uppskatta verksamhetsnyttan genom att ställa kostnader och intäkter mot varandra för en konkret verksamhetssituation (eng. business case).

Introducering av valda verktyg i en organisation kan göras i ett pilotprojekt som gör det möjligt att minimera påverkan och skadan om större hinder skulle upptäckas eller om pilotprojektet misslyckas.

Ett pilotprojekt har följande målsättningar:
- Att skaffa mer detaljerad kunskap om verktyget.
- Utvärdera hur verktyget passar in i existerande processer och arbetssätt samt avgöra vilka eventuella ändringar i dessa som bör göras.
- Att besluta om standardiserade sätt att använda, hantera, lagra och underhålla verktyget och testprodukter (t.ex. besluta om namnkonventioner för filer, tester, skapa bibliotek och definiera testsviters modularitet).
- Att utvärdera om fördelarna har uppnåtts till rimliga kostnader.

Faktorer som ökar chanserna att lyckas med införandet av ett verktyg omfattar bland annat:
- Stegvis införande av verktyget i organisationen.
- Anpassning och förbättring av processer för att stämma med användandet av verktyget.
- Tillhandahålla utbildning och handledning/mentorskap för nya användare.
- Ta fram en användarhandledning för användandet av verktyget
- Införa ett sätt att få återkoppling och lära sig från verktygsanvändandet.
- Övervaka användningen av verktyget och dess fördelar.
- Tillhandahålla stöd till testgruppen för ett utpekat verktyg.
- Samla in inhämtad kunskap från alla grupper.

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

7. Referenser

Standarder

ISTQB Glossary of terms used in Software Testing Version 2.1

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

[IEEE 829-1998] IEEE Std 829™ (1998) IEEE Standard for Software Test Documentation (currently under revision)
Se kapitel 2.3, 2.4, 4.1, 5.2, 5.3, 5.5, 5.6

[IEEE 1028] IEEE Std 1028™ (2008) IEEE Standard for Software Reviews and Audits
Se kapitel 3.2

[IEEE 12207] IEEE 12207/ISO/IEC 12207-2008, Software life cycle processes
Se kapitel 2.1

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

Böcker

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

[Black, 2001] Black, R. (2001) Managing the Testing Process (3rd edition), John Wiley & Sons: New York
Se kapitel 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 kapitel 6.2

[Copeland, 2004] Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: Norwood, MA
Se kapitel 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 kapitel 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 kapitel 6.2, 6.3

[Gilb, 1993]: Gilb, Tom and Graham, Dorothy (1993) Software Inspection, Addison Wesley: Reading, MA
Se kapitel 3.2.2, 3.2.4

[Hetzel, 1988] Hetzel, W. (1988) Complete Guide to Software Testing, QED: Wellesley, MA
Se kapitel 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 Pettticord, B. (2002) Lessons Learned in Software Testing, John Wiley & Sons:
Se kapitel 1.1, 4.5, 5.2

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

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

Bilaga A – Bakgrund till kursplanen

Historien bakom detta dokument

Detta dokument är en svensk översättning av den internationella kursplanen (eng. Syllabus) inklusive anpassning till svensk terminologi. Den första översättningen skedde 2005. Regler för framtagning av kursplanen finns i bilaga C.

Den internationella kursplanen har tagits fram mellan 2004-2011 av en arbetsgrupp bestående av medlemmar utsedda av International Software Testing Qualification Board (ISTQB). Den granskades först av en granskningskommitté och sedan av representanter från programvarutestare från hela världen.
Kursplanen är den som gäller för Certifikat i testning och är den första utbildningsnivån godkänd av ISTQB (**www.istqb.org**
Den svenska kursplanen är en översättning av den internationella Syllabus enligt de regler som gäller för nationell anpassning. I kursplanen har begrepp och termer översatts till svensk terminologi, som också finns sammanställd i tillhörande ordlista.

Målsättningen med testcertifiering

- Att få test erkänt som en viktig och professionell yrkesroll inom programvaruverksamheten.
- Att tillhandahålla ett standardramverk för en karriärsutveckling av testare.
- Att möjliggöra för professionellt kvalificerade testare att bli erkända hos arbetsgivare, kunder och kolleger och att öka testarnas profil.
- Att främja stabila och bra testmetoder inom alla programvarudiscipliner.
- Att identifiera testområden som är av betydelse och har värde för industrin.
- Att göra det möjligt för programvaruleverantörer att anställa certifierade testare och på så sätt få kommersiella fördelar före sina konkurrenter genom att göra reklam för sin anställningspolicy när det gäller testare.
- Att ge testarna och andra med testintresse, en möjlighet i att få en internationellt erkänd examen inom området.

Målsättningen med den internationella examen (tillämpad efter ISTQB-mötet i Sollentuna, November 2001)

- Att göra det möjligt att jämföra testkunskap mellan olika länder.
- Att göra det möjligt för testare att flytta mellan olika länder.
- Att göra det möjligt för multinationella eller internationella projekt att ha en gemensam syn på test och testrelaterade områden.
- Att öka antalet kvalificerade testare över hela världen.
- Att ge ett större mervärde och inverkan genom att bygga på ett internationellt baserat initiativ än att ha ett specifikt angreppssätt per land.
- Att utveckla en gemensam internationell bas av förståelse och kunskap om testning med hjälp av kursplanen och terminologin och att öka förståelsen om testning för alla.
- Att främja testning som ett yrke i fler länder.
- Att göra det möjligt för testare att få en erkänd examen på deras modersmål.
- Att göra det möjligt att utbyta kunskap och resurser mellan länder.
- Att erbjuda ett internationellt erkännande av testare och denna examinering genom att deltagande sker från många länder.

Inträdeskrav för att avlägga denna examen

Inträdeskraven för att ta ett ISTQB-cerifikat i programvarutestning är att kandidaten har ett intresse i programvarutestning. Det är emellertid rekommendabelt att kandidaten också:

- Åtminstone har någon bakgrund i programvaruutveckling eller programvarutestning, t.ex. 6 månaders erfarenhet av systemtest eller användaracceptanstest eller som programvaruutvecklare.
- Deltar i en kurs som har blivit ackrediterad enligt ISTQB-standard (av en av ISTQB godkänd representant för respektive land. När det gäller Sverige är det SSTB)

Bakgrund och historik till certifikat i programvarutestning

Oberoende certifiering av testare började i England med British Computer Society's Information Systems Examination Board (ISEB) när ett råd för programvarutest etablerades 1998 (**www.bcs.org.uk/iseb**). 2002 började ASQF i Tyskland att stödja en tysk testexamenssytem (**www.asqf.de**). Kursplanerna från ISEB och ASQF har varit bas vid framtagning av den internationella ISTQB-Syllabus. Den har sedan omorganiserats, uppdaterats och nya områden har tillförts. Tonvikten är lagd på de områden som ger mest praktiskt stöd åt testarna.

Ett existerande certfikat för programvarutestning (t.ex. från ISEB, ASQF eller ett nationellt ISTQB-organ), som har utgivits innan detta internationella certifikat utgavs, anses vara likvärdigt med det internationella certifikatet. Certifikatet har ingen tidsbegränsning och behöver inte förnyas. Datum för utfärdande finns noterat i certifikatet.

De landslokala aspekterna kontrolleras av respektive organ som är godkänt av ISTQB. Uppgifterna för ett nationellt organ har definierats av ISTQB men har förverkligats i respektive land. Uppgifterna för varje lands organ förväntas vara ackreditering av kursleverantörer och etablerande av examen.

Bilaga B – Inlärningsmål/ kunskapsnivå

Denna kursplan använder följande nivåer för att definiera inlärningsmål. För varje modul beskrivs inlärningsmålen med hjälp av dessa nivåer i inledningen av respektive kapitel.

Nivå 1: Att komma ihåg (K1)

Kandidaten skall känna igen, minnas och erinra sig ett uttryck eller ett begrepp.

Nyckelord: Minnas, erinra sig, känna igen, kunna.

Exempel

Känna igen definitionen av ”felsymptom” såsom

- “utebliven leverans av en tjänst till en slutanvändare eller annan intressent” eller
- ”avvikelse från programvarans eller systemets förväntade beteende eller resultat”

Nivå 2: Att förstå (K2)

Kandidaten kan motivera påståenden inom ämnesområdet med anledningar eller förklaringar. Deltagaren kan summera, särskilja, klassificera och exemplifiera fakta (t.ex. jämföra begrepp), testprinciper och testprocedurer (t.ex. förklara aktivitetsordningen).

Nyckelord: Sammanfatta, klassificera, jämföra, kartlägga, kontrastera exemplifiera, förklara, tolka, beskriva dra slutsatser, summera, kategorisera.

Exempel

Förklara varför testfall skall designas så tidigt som möjligt.

- För att upptäcka defekter när det är billigt att eliminera dem.
- För att upptäcka de viktigaste defekterna först.

Förklara likheter och skillnader mellan integrations- och systemtestning.

- Likheter: mer än en komponent testas, icke-funktionella egenskaper kan testas.
- Olikheter: integrationstestning är inriktad på gränssnitt och samspel, medan systemtestning är inriktad på helheten såsom testning utgående från ett användarperspektiv.

Nivå 3: Tillämpa (K3)

Kandidaten kan välja rätt användning av ett koncept eller en teknik och tillämpa den utifrån ett givet sammanhang. K3 gäller oftast för kunskaper om procedurer. K3 kräver ingen kreativitet, som i fallet med utvärdering av en programvaruapplikation eller skapandet av en modell för en given programvara. K3 är t.ex. när vi utifrån en given modell beskriver de olika stegen för att skapa testfall.

Nyckelord: Implementera, exekvera, använda, följa en metod, tillämpa en metod

Exempel

- Kunna identifiera gränsvärden för partitioner (eller klasser) med giltiga eller ogiltiga värden.
- Kunna, med hjälp av ett tillståndsdiagram och en uppsättning existerande testfall, använda den generiska metoden tillståndsbaserad testning.

Nivå 4: Analysera (K4)

Kandidaten kan dela upp information som är relaterad till en metod eller en teknik i dess beståndsdelar för en ökad förståelse, och kan skilja på fakta och slutsatser. Typiska användningsområden är att analysera dokument, programvara eller projektläge och föreslå lämpliga åtgärder för att lösa problem eller genomföra arbetsuppgifter.

Nyckelord: Analysera, differentiera, välja, strukturera, fokusera, beskriva attribut, dela upp, utvärdera, bedöma, övervaka, koordinera, skapa, syntetisera, generera, göra antaganden, planera, designa, konstruera, producera.

Exempel

- Kunna analysera produktrisker och föreslå förebyggande och korrigerande åtgärder för att begränsa riskerna
- Kunna beskriva vilka delar i en avvikelserapport som är fakta och vad som är slutsatser dragna utifrån

Referenser (för inlärningsmålens kognitiva nivåer)

Bloom, B. S. (1956). *Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain,* David McKay, Co. Inc.
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.

Bilaga C – Regler som tillämpas av ISTQB/ SSTB för den svenska kursplanen

Reglerna nedan användes vid framtagning och granskning av den svenska översättningen. För regler som användes vid framtagning av den internationella syllabus, se appendix C hos denna.

Generella regler

GR1. Kursplanen är avsedd för följande läsare:

- Kursleverantörer, för att sätta upp en kurs.
- Det svenska ackrediteringsorganet, SSTB för att ackreditera kursleverantörer enligt ISTQB
- Det svenska examineringsorganet, SSTB för att skapa examineringsfrågor och genomföra examen.

Det förutsätts att ovannämnda har en kunskap om test för att motsvarande denna nivå samt flera års erfarenhet av testarbete.

Kursplanen avser att resultera i ISTQB internationellt godkänd kurs genom sin svenska organisation SSTB, där kandidaterna ej behöver tidigare erfarenhet av testning.

GR2. Kursplanen skall vara mer praktisk inriktad än teoretisk.

GR3. Kursplanen skall vara tydlig och entydig för dess avsedda läsare enligt GR1.

Aktuellt innehåll

AI1. Kursplanen skall innehålla aktuella testbegrepp och skall återspegla best practice inom programtestning där det är erkänt. Kursplanen skall granskas var tredje till femte år.

AI2. Tidsrelaterade ämnen skall minimeras, t.ex. dagens marknadsläge, för att säkra att dokumentet har en livslängd på tre till fem år.

Inlärningsmål

IM1. Inlärningsmålen skall skilja mellan områden

- där kandidaten skall känna igen eller komma ihåg (K1),
- där kandidaten begreppsmässigt skall förstå (K2)
- där kandidaten skall kunna tillämpa, (K3)
- där kandidaten skall kunna analysera ett dokument, en programvara och/eller en projektsituation i ett givet sammanhang (K4)

IM2. Innehållsbeskrivningen skall överensstämma med inlärningsmålen.

IM3. För att illustrera inlärningsmålen skall ett urval av examinationsfrågor ges ut tillsammans med kursplanen.

Helhetsstruktur

ST1. Strukturen hos kursplanen skall vara tydlig och tillåta korsreferenser till och från andra delar, från examinationsfrågor och från andra relevanta dokument.

ST2. Överlapp mellan kapitel i kursplanen skall vara minimalt.

ST3. Alla kapitel i kursplanen skall ha samma struktur.

ST4. Kursplanen skall innehålla version, utgivningsdatum och sidnummer på varje sida.

ST5. Kursplanen skall innehålla riktlinjer för hur mycket tid som skall ägnas åt varje kapitel (för att spegla den relativa viktigheten av varje ämne).

Referenser

SR1. Informationskällor och referenser skall ges för begrepp som används i kursplanen för att hjälpa kursleverantörer att hitta mer information om området.

SR2. Där det inte finns tydligt klara och identifierade källor, skall det finnas mer information i kursplanen. Detta görs på så sätt att begreppen listas i kursplanen medan definitionerna återfinns i ordlistan.

Informationskällor

Begrepp som används kursplanen är definierade i SSTBs ordlista som också innehåller en referens till begrepp använda i standarder.

En lista på rekommenderade böcker om test finns tillsammans med denna kursplan. Denna finns under avsnittet referenser. Dessa böcker är hittills ej översatta till svenska.

Bilaga D – Information till kursleverantörer

Varje avsnittsrubrik i kursplanen har tilldelats en tid i minuter. Syftet med detta är:

- Att ge rekommendation om tiden för detta avsnitt relaterat till andra avsnitt i kursplanen.
- Att ge en ungefärlig minimitid för att lära ut varje avsnitt.

Kursleverantören kan använda mer tid än vad som indikeras och eleven kan använda ännu mer tid till att läsa och undersöka.

Ett kursupplägg behöver inte följa samma ordning som denna kursplan indikerar.

Kursplanen innehåller referenser till etablerade standarder som måste användas vid förberedelse av kursmaterialet. Varje standard som används måste ha samma version som specificerats i kursplanen. Andra publikationer, mallar eller standarder som inte refereras till i denna kursplan får användas eller refereras till, men de ingår inte examinationsunderlaget.

Speciella områden i kursplanen som kräver praktiska övningar är följande:

4.3 Specifikationsbaserade eller black-box-tekniker

Praktiskt arbete (korta övningar) skall inkluderas och täcka de fyra teknikerna: ekvivalensklasser, gränsvärdesanalys, testning med hjälp av beslutstabeller och tillståndsbaserad testning. Lektionerna och övningarna för dessa tekniker skall baseras på referenser för respektive teknik.

4.4 Strukturbaserade eller white-box-tekniker

Praktiskt arbete (korta övningar) skall inkluderas för att utvärdera om en grupp av tester har uppnått 100 % satstäckning och 100 % beslutstäckning, men också för att konstruera testfall för ett givet styrflöde.

5.6 Avvikelsehantering

Praktiskt arbete (korta övningar) skall inkluderas för att skriva och/eller utvärdera en avvikelserapport

Bilaga E – Ändringar i Kursplan 2011

Alla ändringar gjorda i denna kursplan 2011 finns beskrivna i Kursplan_2011_ändringar.

Index