Certified Tester

Foundation Level
Kursplan

Version 2018

Swedish Software Testing Board

International Software Testing Qualifications Board

Revisionshistorik

Version	Datum	Kommentarer
SSTB 2005-2006 (1.1)	2006-03-07	Första svenska utgåvan
SSTB 2007 (2.0)	2007-12-20	Versionsnummer ändrat till 2.0 (internt) Dokumentnamn ändrat till version 2007 (överensstämmer med engelska versionen) Åtgärdat inkomna kommentarer. Harmonisering med ”Syllabus CTFL ver. 2007”.
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
SSTB 2018	2018-11-01	Uppdaterad efter ISTQB CTFL Syllabus 2018

Innehållsförteckning

Tillkännagivanden
Det engelska originalet till detta dokument har producerats av ISTQB:s arbetsgrupp för grundnivån (version 2018): Klaus Olsen (ordförande), Tauhida Parveen (vice ordförande), Rex Black (projektledare), Debra Friedenberg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh och Eshraka Zakari samt alla nationella styrelser med deras förslag.

0 Inledning

0.1 Syfte med kursplanen

Den här kursplanen utgör grunden för certifiering på ISTQB Foundation Level. SSTB tillhandahåller kursplanen i följande syften:

1. Till kurshållare för att ta fram kursmaterial och som vägledning för att hitta lämpliga undervisningsmetoder.
2. Till de som vill certifiera sig så att de kan förbereda sig för certifieringsexamineringen (antingen som del av en utbildning eller fristående).
3. Till programvaru- och systemtekniker som ett hjälpmedel för att öka den samlade kunskapen om programvaru- och systemtestning, och som grund för böcker och artiklar.

ISTQB och SSTB kan också tillåta andra organ och enskilda att använda kursplanen i andra syften förutsatt att de inhämtar skriftligt godkännande från ISTQB och SSTB i förväg.

0.2 Certifierad testare på grundnivå för programvarutestning

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

ISTQB Foundation Level Overview 2018 är ett separat dokument som innehåller följande information:

- Verksamhetsresultat (business outcomes) för kursplanen
- Matris som visar spårbarhet mellan verksamhetsresultaten och utbildningsmålen
- Sammanfattning av den här kursplanen

0.3 Utbildningsmål som kan examineras och kognitiva kunskapsnivåer

Utbildningsmålen ligger till grund för verksamhetsresultaten och används för att skapa de examineringar som används för att certifiera testare på Foundation Level.

Rent allmänt kan allt innehåll i den här kursplanen examineras på K1-nivå, utom inledningen och bilagorna. Detta innebär att kandidaten kan uppmanas att identifiera, komma ihåg eller minnas ett nyckelord eller koncept som nämns i något av de sex kapitlen. Kunskapsnivåerna för de specifika utbildningsmålen visas i början av varje kapitel och klassificeras enligt följande:

- K1: komma ihåg
- K2: förstå
- K3: tillämpa

Mer information och exempel på utbildningsmål finns i bilaga B.

Definitionerna för alla termer som listas som nyckelord strax under kapitelrubrikerna ska kommas ihåg (K1), även om de inte uttryckligen nämns i utbildningsmålen.

0.4 Certifieringsexaminering på Foundation Level

Certifieringsexamineringen för Foundation Level bygger på den här kursplanen. Svaren på examineringsfrågorna kan kräva användning av material som bygger på mer än ett avsnitt i kursplanen. Alla avsnitt i kursplanen kan examineras, utom inledningen och bilagorna. Standarder, böcker och andra ISTQB-kursplaner inkluderas som referenser, men innehållet i sådana standarder, böcker och andra ISTQB-kursplaner är inte att examinerbara utöver vad som sammanfattas i den här kursplanen.

Examineringen sker i form av flervalsfrågor. Det finns 40 frågor. För att klara examineringen måste minst 65 % av frågorna (det vill säga 26 frågor) besvaras korrekt.

Examineringen kan genomföras som del av en ackrediterad kurs eller fristående (till exempel vid ett examinationscenter eller genom en öppen (publik) examinering). Deltagande i en ackrediterad kurs är ingen förutsättning för examineringen.

0.5 Ackreditering

Ett nationellt organ inom ISTQB får ackreditera kurshållare vars kursmaterial följer den här kursplanen. Kurshållare ska inhämta riktlinjer för ackreditering från det nationella organet eller det organ som utför ackrediteringen. En ackrediterad kurs anses följa den här kursplanen och får ha en ISTQB-examinering i anslutning till kursen.

0.6 Detaljnivå

Detaljnivån i syllabus och i den här kursplanen ger möjlighet till att skapa kurser och examineringar som stämmer överens internationellt. För att uppnå det målet innehåller kursplanen:

- Allmänna instruktionsmål som beskriver syftet med Foundation Level
- En lista med termer som eleverna måste komma ihåg
- Utbildningsmål för varje kunskapsområde som beskriver det kognitiva utbildningsresultat som ska uppnås
- En beskrivning av viktiga koncept, inklusive referenser till källor som godkänd litteratur eller standarder

Kursplanens innehåll är inte en beskrivning av hela kunskapsområdet för programvarutestning, utan återspeglar i stället den detaljnivå som ska ingå i utbildningar på grundnivå. Kursplanen fokuserar på testkoncept och tekniker som kan tillämpas för alla programvaruprojekt, inklusive agila projekt. Den här kursplanen innehåller inga specifika utbildningsmål som rör en viss metod eller utvecklingslivscykel för programvaran, men den tar upp hur dessa koncept kan tillämpas i agila projekt, andra typer av iterativa och inkrementella livscykler och i sekventiella livscykler.

0.7 Kursplanens organisation

Det finns sex kapitel med innehåll som kan examineras. Rubriken på toppnivå för varje kapitel anger tiden för kapitlet. Inga tider anges på lägre kapitelnivåer. För ackrediterade utbildningar kräver kursplanen minst 16,75 timmars undervisning som fördelas över de sex kapitlen enligt följande:

- Kapitel 1: 175 minuter Grunderna inom test
- Kapitel 2: 100 minuter Testning under programvarans utvecklingslivscykel
- Kapitel 3: 135 minuter Statisk testning
- Kapitel 4: 330 minuter Testtekniker
- Kapitel 5: 225 minuter Testledning
- Kapitel 6: 40 minuter Verktygsstöd för testning

1 Grunderna inom test — 175 minuter

Nyckelord

debugging, defekt, felsymptom, grundorsak, kvalitet, kvalitetssäkring, misstag, spårbarhet, testanalys, testavslut, testbas, testdata, testdesign, testfall, testexekvering, testexekveringsschema, testimplementation, testmål, testning, testobjekt, testorakel, testplanering, testprocedur, teststyrning, testsvit, testvara, testvillkor, testövervakning, täckningsgrad, validering, verifiering

Utbildningsmål för grunderna inom test:

1.1 Vad är testning?

FL-1.1.1 (K1) Identifiera typiska mål för testning

FL-1.1.2 (K2) Skilja på testning och debugging

1.2 Varför är testning nödvändigt?

FL-1.2.1 (K2) Ge exempel på varför testning är nödvändigt

FL-1.2.2 (K2) Beskriv relationen mellan testning och kvalitetssäkring och ge exempel på hur testning bidrar till högre kvalitet

FL-1.2.3 (K2) Skilja mellan misstag, defekt och felsymptom

FL-1.2.4 (K2) Skilja mellan grundorsaken till en defekt och dess effekter

1.3 Sju testprinciper

FL-1.3.1 (K2) Förklara de sju testprinciperna

1.4 Testprocess

FL-1.4.1 (K2) Förklara sammanhangets inverkan på testprocessen

FL-1.4.2 (K2) Beskriv testaktiviteterna och de olika uppgifterna som ingår i testprocessen

FL-1.4.3 (K2) Skilja på de arbetsprodukter som ger stöd åt testprocessen

FL-1.4.4 (K2) Förklara värdet av att bibehålla spårbarheten mellan testbasen och testarbetsprodukterna

1.5 Testningens psykologi

FL-1.5.1 (K1) Identifiera de psykologiska faktorer som påverkar resultatet av testningen

FL-1.5.2 (K2) Förklara skillnaden mellan det tankesätt som krävs för testaktiviteter och det tankesätt som krävs för utvecklingsaktiviteter

1.1 Vad är testning?

Programvarusystem är en viktig del av vardagen, från affärsapplikationer (till exempel banktjänster) till konsumentprodukter (till exempel bilar). De flesta av oss har upplevt att programvara inte fungerar som förväntat. Programvara som inte fungerar på rätt sätt kan leda till många problem, inklusive förlust av pengar, tid, affärsrykte och till och med personskador eller dödsfall. Programvarutestning är ett sätt att utvärdera programvarans kvalitet och minska risken för felsymptom under användning av programvaran.

En vanlig missuppfattning om testning är att den endast består av att exekvera tester, det vill säga att köra programvaran och kontrollera resultaten. Enligt beskrivningen i avsnitt 1.4 är programvarutestning en process som omfattar många olika aktiviteter, och testexekvering (inklusive kontroll av resultaten) är bara en av dessa aktiviteter. Testprocessen omfattar också aktiviteter som testplanering, analys, design och implementering av tester, rapportering av teststatus och testresultat samt utvärdering av testobjektets kvalitet.

En del testning omfattar exekvering av den komponent eller det system som testas. Sådan testning kallas dynamisk testning. Annan testning omfattar inte exekvering av den komponent eller det system som testas. Sådan testning kallas statisk testning. Testningen omfattar alltså även granskning av arbetsprodukter som krav, användarberättelser och källkod.

En annan vanlig missuppfattning om testning är att den enbart fokuserar på verifiering av krav, användarberättelser eller andra specifikationer. Även om testningen omfattar att kontrollera om systemet uppfyller de specifika kraven, inkluderar den också validering. Detta innebär att kontrollera om systemet kommer att uppfylla behoven hos användare och andra intressenter i systemets driftsmiljö(er).

Testaktiviteterna organiseras och genomförs på olika sätt i olika livscykler (se avsnitt 2.1).

1.1.1 Typiska testmål

För varje projekt kan testmålen omfatta att:

- Utvärdera arbetsprodukter som krav, användarberättelser, design och kod
- Verifiera att alla specificerade krav är uppfyllda
- Validera om testobjektet är färdigt och fungerar som användaren och andra intressenter förväntar sig
- Skapa förtroende för testobjektets kvalitetsnivå
- Förhindra defekter
- Upptäcka felsymptom och defekter
- Tillhandahålla tillräckligt med information till intressenterna så att de kan fatta välgrundade beslut, särskilt när det gäller testobjektets kvalitetsnivå
- Minska risken för bristfällig programvarukvalitet (till exempel tidigare oupptäckta felsymptom som uppstått under drift)
- Säkerställa efterlevnad av kontraktsmässiga, legala eller reglerande krav eller standarder, och/eller att verifiera testobjektets efterlevnad av sådana krav eller standarder

Målen med testningen kan variera beroende på sammanhanget för komponenten eller systemet som testas, testnivån och programvarans utvecklingslivscykelmodell. Dessa skillnader kan till exempel omfatta:

- Under komponenttestning kan ett mål vara att hitta så många felsymptom som möjligt, så att de bakomliggande defekterna kan identifieras och åtgärdas tidigt. Ett annat mål kan vara att öka kodtäckningen för komponenttesterna.
- Under acceptanstestning kan ett mål vara att bekräfta att systemet fungerar som förväntat och uppfyller kraven. Ett annat mål med den här testningen kan vara att lämna information till intressenterna om riskerna med att leverera systemet vid en viss tidpunkt.

1.1.2 Testning och debugging

Testning och debugging är inte samma sak. När testerna exekveras kan de visa felsymptom som orsakas av defekter i programvaran. Debugging är en utvecklingsaktivitet som går ut på att hitta, analysera och rätta till sådana defekter. Efterföljande omtestning kontrollerar om rättningarna har åtgärdat defekterna. I en del fall ansvarar testarna för det inledande testet och den slutliga omtestningen, medan utvecklarna sköter debuggingen och tillhörande komponenttestning. Vid agil utveckling och i vissa andra livscykler kan testarna också vara delaktiga i debugging och komponenttestning.

ISO-standarden (ISO/IEC/IEEE 29119-1) innehåller ytterligare information om koncept för programvarutestning.

1.2 Varför är testning nödvändigt?

Noggrann testning av komponenter och system samt tillhörande dokumentation kan bidra till att riskerna för felsymptom under drift minskas. När defekter upptäcks och sedan rättas till, bidrar detta till kvaliteten på komponenter eller system. Dessutom kan testning krävas för att uppfylla kontraktsmässiga eller legala krav eller branschspecifika standarder.

1.2.1 Testningens bidrag till ett lyckat resultat

Under hela datoriseringens historia är det relativt vanligt att programvara och system, efter leverans och driftsättning, ger upphov till felsymptom (på grund av defekter) eller på annat sätt inte klarar att uppfylla intressenternas behov. Med hjälp av lämpliga testtekniker är det dock möjligt att minska frekvensen av sådana problematiska leveranser, förutsatt att dessa tekniker används med rätt kunskapsnivå om testning, på lämpliga testnivåer och vid lämpliga tidpunkter under programvarans utvecklingslivscykel. Här är några exempel:

- När testare är delaktiga i att granska kraven eller förfina användarberättelser kan det leda till att defekter upptäcks i arbetsprodukterna. Att identifiera och avlägsna defekter som rör kraven kan minska riskerna för att utveckla felaktiga funktioner eller funktioner som inte går att testa.
- Genom att låta testarna samarbeta nära systemdesigners under utformningen av systemet, kan de olika parternas förståelse för designen och hur den bör testas öka. Den här ökade förståelsen kan minska risken för grundläggande defekter i designen och göra det möjligt att identifiera tester i ett tidigt skede.
- Genom att låta testarna samarbeta nära utvecklarna och programmerarna under utvecklingen, kan de olika parternas förståelse för koden och hur den bör testas öka. Denna ökade förståelse kan minska risken för defekter i koden och testerna.
- Genom att låta testarna verifiera och validera programvaran innan release är det möjligt att upptäcka felsymptom som annars kunde ha missats, samt att stötta processen genom att avlägsna de defekter som orsakar felsymptomen (det vill säga debugging). Detta ökar sannolikheten för att programvaran ska uppfylla intressenternas behov och uppfyller kraven.

Förutom dessa exempel bidrar uppfyllandet av definierade testmål (se avsnitt 1.1.1) till att den övergripande utvecklingen och underhållet av programvaran blir lyckad.

1.2.2 Kvalitetssäkring och testning

Även om många använder termen *kvalitetssäkring* (eller *QA,* Quality Assurance) om testning, är kvalitetssäkring och testning inte samma sak. De är dock besläktade och knyts samman genom ett större koncept, kvalitetsstyrning. Kvalitetsstyrning omfattar alla aktiviteter som syftar till att leda och styra kvaliteten i en organisation. Kvalitetsstyrning omfattar bland annat både kvalitetssäkring och kvalitetskontroll. Kvalitetssäkring fokuserar normalt sett på att följa lämpliga processer så att rätt kvalitetsnivåer går att uppnå. När processerna genomförs på rätt sätt blir de arbetsprodukter som skapas ofta av högre kvalitet, vilket bidrar till att förebygga defekter. Dessutom är grundorsaksanalyser för att upptäcka och åtgärda orsaken till defekterna, tillsammans med korrekt tillämpning av resultaten från retrospektivmöten för att förbättra processerna, viktiga för effektiv kvalitetssäkring.

Kvalitetskontroll omfattar olika aktiviteter, inklusive testaktiviteter, som syftar till att uppnå lämpliga kvalitetsnivåer. Testaktiviteterna är del av den övergripande utvecklings- eller underhållsprocessen för programvaran. Eftersom kvalitetssäkring handlar om korrekt genomförande av hela processen, ger kvalitetssäkring stöd åt korrekt testning. Enligt beskrivningen i avsnitt 1.1.1 och 1.2.1 bidrar testningen till att uppnå kvalitet på flera olika sätt.

1.2.3 Misstag, defekter och felsymptom

En person kan göra ett misstag som kan leda till att det införs en defekt (fel eller bugg) i programvarans kod eller någon annan relaterad arbetsprodukt. Ett misstag som leder till att det införs en defekt i en arbetsprodukt kan utlösa ett misstag som leder till att det införs en defekt även i en relaterad arbetsprodukt. Ett misstag i kraveliciteringen kan leda till en kravdefekt, vilket i sin tur leder till ett programmeringsfel som leder till en defekt i koden.

Om en defekt i koden exekveras kan det leda till ett felsymptom, men inte nödvändigtvis under alla förhållanden. Vissa defekter kan till exempel kräva mycket specificerade indata eller förutsättningar för att orsaka ett felsymptom, som kan uppstå sällan eller aldrig.

Misstag kan uppstå av många orsaker, till exempel:

- Tidsbrist
- Den mänskliga faktorn
- Projektdeltagare som är oerfarna eller saknar nödvändiga färdigheter
- Kommunikationsproblem mellan projektets deltagare, inklusive kommunikationsproblem kring krav och design
- Komplexiteten hos kod, design, arkitektur, det underliggande problemet som ska lösas och/eller den teknik som används
- Missförstånd kring gränssnitt inom och mellan system, särskilt när det finns många interaktioner inom eller mellan systemen
- Nya, obekanta tekniker

Förutom felsymptom som uppstår på grund av defekter i koden, kan felsymptom också orsakas av miljömässiga orsaker. Strålning, elektromagnetiska fält och föroreningar är exempel på saker som kan orsaka defekter i firmware eller påverka exekvering av programvaran genom att ändra villkoren för hårdvaran.

Inte alla testresultat som avviker från det förväntade är felsymptom. Falska positiva resultat kan uppstå därför att testerna genomförs på felaktigt sätt, på grund av defekter i testdata, testmiljö och testvara eller av andra orsaker. Motsatt situation kan också uppstå, där liknande misstag eller defekter leder till falska negativa resultat. Falska negativa resultat är tester som inte upptäcker defekter som de borde ha upptäckt. Falska positiva resultat rapporteras som defekter, men är egentligen inte defekter.

1.2.4 Defekter, grundorsaker och effekter

Grundorsakerna till defekter är de tidigaste handlingarna eller omständigheterna som bidragit till att skapa defekterna. Defekter kan analyseras för att identifiera deras grundorsaker och därmed minska förekomsten av liknande defekter i framtiden. Genom att fokusera på de grundorsaker som är viktigast kan en grundorsaksanalys leda till processförbättringar som i framtiden förhindrar att ett stort antal defekter introduceras.

Antag till exempel att felaktiga räntebetalningar som beror på en enda rad med felaktig kod resulterar i klagomål från kunderna. Den defekta koden skrevs för en användarberättelse som var otydlig på grund av produktägarens bristande kunskaper om ränteberäkning. Om en stor procentandel defekter förekommer i ränteberäkningarna och dessa defekter har sin grundorsak i liknande missförstånd, kan produktägarna få utbildning i ränteberäkning för att minska antalet sådana defekter i framtiden.

I det här fallet utgör klagomål från kunderna effekten. De felaktiga räntebetalningarna är felsymptom. Den felaktiga beräkningen i koden är en defekt som har uppstått på grund av den ursprungliga defekten, som var den otydliga användarberättelsen. Grundorsaken till den ursprungliga defekten var brist på kunskap hos produktägaren, vilket resulterade i att produktägaren gjorde ett misstag när användarberättelsen skrevs. Processen för analys av grundorsaken behandlas i ISTQB-ETM Expert Level Test Management Syllabus och ISTQB-EITP Expert Level Improving the Test Process Syllabus.

1.3 Sju testprinciper

Ett antal testprinciper har föreslagits under de senaste 50 åren och erbjuder allmänna riktlinjer som är gemensamma för all testning.

1. Test visar närvaron av fel men inte frånvaron

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

2. Uttömmande testning är omöjligt

Att testa allt (alla kombinationer av indata och förutsättningar) är inte möjligt annat än i mycket enkla fall. I stället för uttömmande testning bör riskanalyser, testtekniker och prioriteringar användas för att fokusera testinsatserna.

3. Tidig testning sparar tid och pengar

För att hitta defekter tidigt bör både statiska och dynamiska testaktiviteter inledas så tidigt som möjligt under programvarans utvecklingslivscykel. Tidig testning kallas ibland för *shift left*. Tidig testning under programvarans utvecklingslivscykel bidrar till att minska eller eliminera dyrbara ändringar (se avsnitt 3.1).

4. Ansamlingar av fel

Ett litet antal moduler innehåller ofta de flesta av de fel som upptäcks vid testning av en förhandsrelease, eller ger upphov till flest felsymptom i drift. Uppskattade grupper eller ansamlingar av fel, samt de faktiska grupper som observeras under testning eller drift, utgör en viktig del av den riskanalys som används för att fokusera testinsatsen (enligt princip 2).

5. Se upp för immunitetsparadoxen

Om samma tester upprepas gång på gång kommer samma uppsättning tester till slut inte upptäcka några nya defekter. För att upptäcka nya fel kan befintliga tester och testdata behöva ändras, och nya tester kan behöva skapas. (Testerna är inte längre effektiva när det gäller att hitta fel.) I vissa fall, till exempel vid automatiserad regressionstestning, leder immunitetsparadoxen till ett gynnsamt resultat, nämligen ett relativt lågt antal fel på grund av regression.

6. Testning beror på sammanhang

Testning sker på olika sätt i olika sammanhang. Till exempel testas säkerhetskritisk programvara för industrin på annat sätt än en e-handelsapp för mobilen. Ett annat exempel är att testning i ett agilt projekt sker på annat sätt än testning i ett projekt med sekventiell livscykel (se avsnitt 2.1).

7. Frånvaro-av-fel-fallgropen

En del organisationer förväntar sig att testare ska kunna köra alla möjliga tester och hitta alla möjliga fel, men princip 2 respektive 1 visar att det är omöjligt. Dessutom är det en villfarelse att förvänta sig att *bara* hitta och åtgärda ett stort antal fel kan säkerställa att ett system blir bra. Till exempel kan noggrann testning av alla specificerade krav och åtgärdande av alla fel som upptäcks fortfarande leda till ett system som är svårt att använda, inte uppfyller användarens behov och förväntningar eller är sämre än andra konkurrerande system.

Se Myers 2011, Kaner 2002 och Weinberg 2008 för exempel på dessa och andra testprinciper.

1.4 Testprocess

Det finns ingen universell testprocess för programvara, men det finns vanliga uppsättningar testaktiviteter som gör att testning har större chans att uppnå de fastställda målen. Dessa uppsättningar med testaktiviteter kallas för en testprocess. Vilken specifik testprocess som är rätt att använda i en viss situation beror på många faktorer. Vilka testaktiviteter som ingår i testprocessen, hur dessa aktiviteter implementeras och när dessa aktiviteter inträffar kan definieras i organisationens teststrategi.

1.4.1 Testprocessen i ett sammanhang

Sammanhangsberoende faktorer som påverkar testprocessen för en organisation inkluderar bland annat:

- Programvarans utvecklingslivscykelmodell och de projektmetoder som används
- Testnivåer och testtyper att ta hänsyn till
- Produkt- och projektrisker
- Verksamhetsdomän
- Driftbegränsningar, bland annat:
    - Budget och resurser
    - Tidsplaner
    - Komplexitet
    - Kontraktsmässiga och reglerande krav
- Policy och arbetssätt
- Beslutade interna och externa standarder

I följande avsnitt beskrivs allmänna aspekter på organisatoriska testprocesser avseende:

- Testaktiviteter och uppgifter
- Testarbetsprodukter
- Spårbarhet mellan testbasen och testarbetsprodukterna

Det är värdefullt om testbasen för de aktuella testnivåerna eller testtyperna har definierade mätbara kriterier för täckningsgraden. Kriterierna för täckningsgraden kan i praktiken fungera som nyckeltal (KPI:er, Key Performance Indicator) för att driva de aktiviteter som visar att testmålet för programvaran har uppnåtts (se avsnitt 1.1.1).

Till exempel kan testbasen för en mobilapplikation omfatta en lista med krav och en lista med mobila enheter som stöds. Varje krav är ett element i testbasen. Varje enhet som stöds är också ett element i testbasen. Kriterierna för täckningsgraden kan kräva minst ett testfall för varje element i testbasen. Efter exekveringen kan resultatet av dessa tester visa intressenterna om de angivna kraven är uppfyllda och om felsymptom observerades på de enheter som stöds.

ISO-standarden (ISO/IEC/IEEE 29119-2) innehåller ytterligare information om testprocesser.

1.4.2 Testaktiviteter och uppgifter

En testprocess består av följande huvudsakliga aktivitetsgrupper:

- Testplanering
- Testövervakning och teststyrning
- Testanalys
- Testdesign
- Testimplementation
- Testexekvering
- Testavslut

En aktivitetsgrupp består av aktiviteter, som beskrivs nedan. Varje aktivitet kan i sin tur bestå av flera individuella uppgifter, som varierar mellan olika projekt eller releaser.

Även om många av dessa aktivitetsgrupper logiskt sett kan verka vara sekventiella, implementeras de oftast iterativt. Agil utveckling omfattar till exempel små iterationer av programvarudesign, byggen och testning som sker kontinuerligt, med stöd av pågående planering. Därför inträffar testaktiviteterna iterativt och kontinuerligt inom detta utvecklingsarbetssättet. Även under sekventiell utveckling inkluderar den stegvisa logiska sekvensen överlappning, kombinationer, parallellitet eller utelämnande, så att det ofta blir nödvändigt att skräddarsy dessa huvudaktiviteter efter systemets sammanhang och det aktuella projektet.

Testplanering

Testplanering omfattar aktiviteter som definierar testmålen och metoden för att uppnå dessa inom de begränsningar som anges av sammanhanget (till exempel att specificera lämpliga testtekniker och uppgifter samt att ta fram ett testschema för att uppfylla en deadline). Testplanerna kan revideras baserat på återkoppling från aktiviteterna under testövervakning och teststyrning. Testplanering beskrivs vidare i avsnitt 5.2.

Testövervakning och teststyrning

Testövervakning omfattar pågående jämförelser av verkliga framsteg jämfört med testplanen med hjälp av de mätetal för testövervakning som definierats i testplanen. Teststyrning omfattar nödvändiga åtgärder för att uppfylla målen i testplanen (som kan uppdateras över tiden). Testövervakning och teststyrning får

stöd genom utvärderingen av avslutskriterier, som hänvisas till ”definition of done” i vissa livscykler (se ISTQB-AT Foundation Level Agile Tester Extension Syllabus). Utvärderingen av avslutskriterierna för testexekvering som del av en given testnivå kan till exempel omfatta:

- Kontrollera testresultat och loggar mot angivna kriterier för täckningsgrad
- Utvärdera kvalitetsnivån för en komponent eller ett system baserat på testresultat och loggar
- Fastställa om mer test behövs (om till exempel testerna som ursprungligen skulle uppnå en viss täckningsgrad för produktrisken inte lyckades uppnå målet kan ytterligare tester skrivas och exekveras)

Teststatus jämfört med planen presenteras för intressenterna i teststatusrapporter, inklusive avvikelser från plan och information till stöd för eventuella beslut att stoppa testerna.

Testövervakning och teststyrning beskrivs vidare i avsnitt 5.3.

Testanalys

Under testanalysen analyseras testbasen för att identifiera testbara funktioner och definiera tillhörande testvillkor. Med andra ord fastställer testanalysen ”vad som ska testas” när det gäller mätbara kriterier för täckningsgraden.

Testanalys inkluderar följande större aktiviteter:

- Analys av testbasen för den aktuella testnivån, till exempel:
  - Kravspecifikationer, exempelvis verksamhetskrav, funktionella krav, systemkrav, användarberättelser, epics, användningsfall eller liknande arbetsprodukter som specificerar önskade funktionella eller icke-funktionella komponent- eller systembeteenden
  - Design- och implementationsinformation, exempelvis diagram eller dokument för system- och programvaruarkitektur, designspecifikationer, anropsflöden, modelldiagram (till exempel UML eller relationsdiagram), gränssnittsspecifikationer eller liknande arbetsprodukter som anger komponent- eller systemstrukturen
  - Implementering av komponenter eller själva systemet, inklusive kod, metadata och frågor för databasen samt gränssnitt
  - Riskanalyser, som kan innehålla funktionella, icke-funktionella och strukturella aspekter för komponenter eller system
- Utvärdering av testbasen och testelementen för att identifiera defekter av olika typer, till exempel:
  - Tvetydigheter
  - Utelämnanden
  - Inkonsekvenser
  - Felaktigheter
  - Motsägelser
  - Överflödiga kodsatser
- Identifiera funktioner och funktionsuppsättningar som ska testas

- Definiera och prioritera testvillkor för varje funktion baserat på analys av testbasen, och med tanke på funktionella, icke-funktionella och strukturella egenskaper, andra verksamhetsfaktorer och tekniska faktorer och risknivåer
- Fånga dubbelriktad spårbarhet mellan varje element i testbasen och tillhörande testvillkor (se avsnitten 1.4.3 och 1.4.4)

Black-box-, white-box- och erfarenhetsbaserade testtekniker kan vara användbara under testanalysprocessen (se kapitel 4) för att minska risken att utelämna viktiga testvillkor och definiera mer exakta och noggranna testvillkor.

I vissa fall producerar testanalysen testvillkor som ska användas som testmål i testcharter. Testcharter är typiska arbetsprodukter för vissa typer av erfarenhetsbaserad testning (se avsnitt 4.4.2). När dessa testmål är spårbara till testbasen kan täckningsgraden som uppnås under sådan erfarenhetsbaserad testning mätas.

Identifieringen av defekter under testanalysen är en viktig potentiell fördel, särskilt när inga andra granskningsprocesser används och/eller testprocessen är nära sammanbunden med granskningsprocessen. Sådana testanalysaktiviteter verifierar inte bara om kraven är konsekventa, korrekt uttryckta och kompletta, utan validerar också om kraven på korrekt sätt fångar upp behoven hos kunder, användare och andra intressenter. Ett exempel är att tekniker som beteendedriven utveckling (BDD) och acceptanstestdriven utveckling (ATDD), som omfattar generering av testvillkor och testfall från användarberättelser och acceptanskriterier innan kodningen, också kan användas för att verifiera, validera och upptäcka defekter i användarberättelser och acceptanskriterier (se ISTQB Foundation Level Agile Tester Extension Syllabus).

Testdesign

Under testdesignen utvecklas testvillkoren till högnivåtestfall, uppsättningar med högnivåtestfall och annan testvara. Testanalysen besvarar alltså frågan ”vad ska testas?”, medan testdesign besvarar frågan ”hur ska vi testa?”.

Testdesign omfattar följande större aktiviteter:

- Designa och prioritera testfall och uppsättningar med testfall
- Identifiera nödvändiga testdata till stöd för testvillkor och testfall
- Designa testmiljön och identifiera nödvändig infrastruktur och nödvändiga verktyg
- Fånga dubbelriktad spårbarhet mellan testbasen, testfallen och testprocedurerna (se avsnitt 1.4.4)

Utvecklingen av testvillkor till testfall och uppsättningar med testfall under testdesignen inkluderar oftast användningen av testtekniker (se kapitel 4).

Precis som testanalysen kan alltså testdesign leda till identifiering av liknande typer av defekter i testbasen. Precis som testanalysen är identifiering av defekter under testdesignen en viktig fördel.

Testimplementation

Under testimplementationen skapas och/eller kompletteras den testvara som är nödvändig för testexekveringen, inklusive att ordna testfallen till testprocedurer. Testdesignen besvarar alltså frågan ”hur ska vi testa?”, medan testimplementering besvarar frågan ”har vi nu allt på plats för att köra testerna?”

Testimplementering omfattar följande större aktiviteter:

- Utveckla och prioritera testprocedurer och eventuellt skapa automatiserade testskript

- Skapa testsviter av testprocedurerna och (eventuella) automatiserade testskript
- Ordna testsviterna i ett testexekveringsschema på ett sätt som ger effektiv testexekvering (se avsnitt 5.2.4)
- Bygga testmiljön (som eventuellt omfattar testexekveringsplattformar, tjänstevirtualisering, simulatorer och andra infrastrukturobjekt) och verifiera att allt har ställts in korrekt
- Förbereda testdata och säkerställa att de blir laddade i testmiljön
- Verifiera och uppdatera dubbelriktad spårbarhet mellan testbasen, testvillkoren, testfallen, testprocedurerna och testsviterna (se avsnitt 1.4.4)

Testdesign och testimplementation är uppgifter som ofta kombineras.

I utforskande testning och andra typer av erfarenhetsbaserad testning kan testdesign och implementation ske, och dokumenteras, som del av testexekveringen. Utforskande testning kan bygga på testcharter (framtagna som del av testanalysen), och exekveras omedelbart efter att de har designats och implementerats (se avsnitt 4.4.2).

Testexekvering

Under testexekvering körs testsviter i enlighet med testexekveringsschemat.

Testexekvering inkluderar följande större aktiviteter:

- Registrering av ID och versioner av testelement eller testobjekt, testverktyg och testvara
- Exekvera tester, antingen manuellt eller med testexekveringsverktyg
- Jämföra faktiska resultat med förväntade resultat
- Analysera anomalier för att fastställa sannolika orsaker (till exempel att felsymptom kan uppstå på grund av defekter i koden, men falska positiva resultat kan också uppstå [se avsnitt 1.2.3])
- Rapportera defekter baserat på observerade felsymptom (se avsnitt 5.6)
- Logga resultatet av testexekvering (till exempel godkänd, underkänd, blockerad)
- Repetera testaktiviteter, antingen som resultat av en åtgärd som vidtagits på grund av en anomali, eller som del av den planerade testningen (till exempel körning av ett uppdaterat test, omtestning och/eller regressionstestning)
- Verifiera och uppdatera dubbelriktad spårbarhet mellan testbas, testvillkor, testfall, testprocedurer och testresultat

Testavslut

Aktiviteterna under testavslutet går ut på att samla in data från avslutade testaktiviteter i syfte att konsolidera erfarenheter, testvara och annan relevant information. Testavslutsaktiviteterna inträffar vid projektets milstolpar, till exempel vid release av ett system, ett testprojekt slutförs (eller avbryts), en iteration i ett agilt projekt slutförs (till exempel som del av ett retrospektivmöte), en testnivå slutförs eller en underhållsrelease har slutförts.

Testavslut omfattar följande större aktiviteter:

- Kontroll av att alla felrapporter har avslutats och uppdatera ändringsbegäran eller objekt i produktbackloggen för alla defekter som förblir olösta när testexekveringen är färdig
- Skapa en sammanfattande testrapport för att informera intressenterna

- Slutföra och spara testmiljön, testdata, testinfrastrukturen och annan testvara för senare återanvändning
- Överlämna testvaran till underhållsteamen, andra projektteam och/eller andra intressenter som kan ha nytta av att använda den
- Analysera lärdomar av de slutförda testaktiviteterna för att fastställa vilka ändringar som krävs för framtida iterationer, releaser och projekt
- Använda informationen som samlats in till att förbättra testprocessens mognad

1.4.3 Testarbetsprodukter

Testarbetsprodukter skapas som en del av testprocessen. På samma sätt som det finns stora variationer i hur organisationer implementerar sina testprocesser, finns det också stora variationer i de typer av arbetsprodukter som skapas under processen, hur dessa arbetsprodukter organiseras och hanteras samt vilka namn som används för arbetsprodukterna. Den här kursplanen följer den testprocess som beskrivs ovan samt de arbetsprodukter som beskrivs i denna kursplan samt ISTQB:s ordlista. ISO-standarden (ISO/IEC/IEEE 29119-3) kan också utgöra en riktlinje för testarbetsprodukter.

Många av de testarbetsprodukter som beskrivs i det här avsnittet kan användas och hanteras med testledningsverktyg och felhanteringsverktyg (se kapitel 6).

Arbetsprodukter för testplanering

Arbetsprodukter för testplanering innehåller normalt sett en eller flera testplaner. Testplanen omfattar information om testbasen som de andra testarbetsprodukterna ska kopplas till via spårbarhet (se nedan och avsnitt 1.4.4), samt avslutskriterier (eller ”definition of done”) som ska användas under testövervakning och styrning. Testplanerna beskrivs i avsnitt 5.2.

Arbetsprodukter för testövervakning och -styrning

Arbetsprodukter för testövervakning och -styrning omfattar vanligtvis olika typer av testrapporter, inklusive teststatusrapporter (tas fram löpande och/eller regelbundet) och sammanfattande testrapporter (som tas fram vid olika milstolpar för slutförandet). Alla testrapporter bör ge information som är relevant för mottagarna om teststatus vid rapportdatum, inklusive sammanfattning av testexekveringsresultaten så fort de blir tillgängliga.

Arbetsprodukter för testövervakning och -styrning hanterar också projektledningsfrågor, till exempel uppgiftsstatus, resurstilldelning, resursanvändning och arbetsinsats.

Testövervakning och -styrning samt de arbetsprodukter som skapas under dessa aktiviteter beskrivs ytterligare i avsnitt 5.3 i den här kursplanen.

Arbetsprodukter för testanalys

Arbetsprodukter för testanalys inkluderar definierade och prioriterade testvillkor, som vart och ett helst ska ha dubbelriktad spårbarhet till de specifika elementen i den testbas som villkoret täcker. För utforskande testning kan testanalysen omfatta skapande av testcharter. Testanalysen kan alltså resultera i upptäckt och rapportering av defekter i testbasen.

Arbetsprodukter för testdesign

Testdesign resulterar i testfall och uppsättningar med testfall som används för att exekvera de testvillkor som definieras i testanalysen. Det är ofta bra att ta fram högnivåtestfall utan konkreta värden för indata och förväntade resultat. Sådana högnivåtestfall kan återanvändas under flera testcykler med olika konkreta data, samtidigt som omfattningen av testfallet blir tillräckligt dokumenterad. Helst ska varje testfall vara dubbelriktat spårbart till de testvillkor det täcker.

Testdesign resulterar också i design och/eller identifiering av nödvändiga testdata, design av testmiljö och identifiering av infrastruktur och verktyg, även om det i hög grad varierar i vilken omfattning resultaten dokumenteras.

Testvillkoren som definieras i testanalysen kan förfinas ytterligare i testdesignen.

Arbetsprodukter för testimplementation

Arbetsprodukter för testimplementation omfattar:

- Testprocedurer och ordningsföljden för dessa testprocedurer
- Testsviter
- Ett testexekveringsschema

När testimplementationen är färdig kan de kriterier för täckningsgraden som fastställts i testplanen påvisas genom dubbelriktad spårbarhet mellan testprocedurer och specifika element i testbasen via testfallen och testvillkoren.

I en del fall omfattar testimplementationen att skapa arbetsprodukter som använder eller används av verktyg, till exempel tjänstevirtualisering och automatiserade testskript.

Testimplementation kan också omfatta skapande och verifiering av testdata och testmiljön. Dokumentationen av resultatet från verifiering av data och/eller testmiljön kan variera i hög grad.

Testdata används för att tilldela konkreta värden till indata och de förväntade resultaten av testfallen. Sådana konkreta värden omvandlar, tillsammans med uttryckliga instruktioner om användningen av dem, högnivåtestfall till körbara lågnivåtestfall. Samma högnivåtestfall kan använda olika testdata när det körs med olika releaser av testobjektet. De konkreta förväntade resultaten som förknippas med konkreta testdata identifieras med hjälp av ett testorakel.

Under utforskande testning kan en del arbetsprodukter för testdesign och testimplementation skapas under testexekveringen, även om dokumentationen av den utforskande testningen (och spårbarheten till specifika element i testbasen) kan variera i hög grad.

De testvillkor som definieras i testanalysen kan förfinas ytterligare under testimplementationen.

Arbetsprodukter för testexekvering

Arbetsprodukter för testexekvering omfattar:

- Dokumentation av statusen för individuella testfall eller testprocedurer (till exempel klar för körning, godkänd, underkänd, blockerad, avsiktligt överhoppad o.s.v.)
- Felrapporter (se avsnitt 5.6)
- Dokumentation om vilka testelement, testobjekt, testverktyg och testvara som ingått i testningen

Idealiskt sett kan statusen för varje element i testbasen fastställas och rapporteras efter testexekveringen via dubbelriktad spårbarhet med tillhörande testprocedurer. Vi kan till exempel ange vilka krav som har blivit godkända i alla planerade tester, vilka krav som har tester som inte godkänts och/eller defekter förknippade med sig och vilka krav som har planerade tester som fortfarande väntar på att köras. Detta gör det möjligt att dels verifiera att kriterier för täckningsgraden har uppfyllts, och dels att rapportera testresultat i termer som intressenterna kan förstå.

Arbetsprodukter för testavslut

Arbetsprodukter för testavslut omfattar sammanfattande testrapporter, åtgärdspunkter för att förbättra efterföljande projekt eller iterationer (till exempel efter ett agilt retrospektiv), ändringsbegäran eller objekt i produktbackloggen samt avslutad testvara.

1.4.4 Spårbarhet mellan testbasen och testarbetsprodukterna

Som nämnts i avsnitt 1.4.3 kan testarbetsprodukterna och namnen på dessa variera i hög grad. Oavsett dessa variationer är det viktigt att etablera och bibehålla spårbarhet under testprocessen mellan varje element i testbasen och de olika testarbetsprodukter som är associerade med det aktuella elementet enligt beskrivningen ovan, för att kunna implementera effektiv testövervakning och styrning. Förutom utvärderingen av testtäckningsgraden ger bra spårbarhet stöd för att:

- Analysera ändringarnas påverkan
- Göra testerna möjliga att revidera
- Uppfylla IT-ledningskraven
- Förbättra förståelsen för teststatusrapporter och sammanfattande testrapporter genom att inkludera status för element i testbasen (t.ex. krav vilkas tester blivit godkända, krav vilkas tester inte godkänts och krav vilkas tester väntar på att exekveras)
- Framföra de tekniska aspekterna av testningen till intressenterna i termer som de kan förstå
- Lämna information för utvärdering av produktkvaliteten, processkapaciteten och projektstatusen jämfört med affärsmålen

En del testledningsverktyg tillhandahåller modeller för testarbetsprodukter som matchar alla eller delar av de testprodukter som beskrivs i det här avsnittet. En del organisationer skapar sina egna hanteringssystem för att organisera arbetsprodukterna och tillhandahålla den informationsspårbarhet som krävs.

1.5 Testningens psykologi

Utveckling av programvara, inklusive programvarutestning, utförs av människor. Därför har den mänskliga psykologin stor inverkan på programvarutestning.

1.5.1 Den mänskliga psykologin och testning

Att identifiera defekter under statiska tester som kravgranskning eller vid förfining av användarberättelser, eller att identifiera felsymptom under dynamisk testexekvering, kan uppfattas som kritik av produkten och dess författare. En del av den mänskliga psykologin som kallas bekräftelseförväntan kan göra det svårt att acceptera information som inte stämmer med en persons aktuella uppfattning. Eftersom utvecklarna till exempel förväntar sig att deras kod ska vara korrekt, har de en bekräftelseförväntan som gör det svårt för dem att acceptera att koden är felaktig. Förutom bekräftelseförväntan finns det andra kognitiva förutfattade meningar som gör det svårt för människor att förstå eller acceptera informationen som framkommer genom testningen. Dessutom är det ett vanligt mänskligt drag att skylla på den som kommer med dåliga nyheter, och informationen som uppstår genom testningen innehåller ofta dåliga nyheter.

Som resultat av dessa psykologiska faktorer kan vissa uppfatta testning som en destruktiv aktivitet, trots att den i hög grad bidrar till att föra projektet framåt och öka produktkvaliteten (se avsnitt 1.1 och 1.2). För att försöka minska dessa uppfattningar bör information om defekter och felsymptom framföras på ett konstruktivt sätt. På så vis kan spänningar mellan testare och analytiker, produktägare, designers och utvecklare minskas. Detta gäller för både statisk och dynamisk testning.

Testare och testledare måste ha god social kompetens så att de effektivt kan informera om defekter, felsymptom, testresultat, teststatus och risker, samt skapa positiva relationer med sina kollegor. Här följer några exempel på god kommunikation:

- Börja med att samarbeta i stället för att ta strid. Påminn alla om det gemensamma målet att skapa system av högre kvalitet.
- Understryk fördelarna med testning. För författare kan till exempel informationen hjälpa dem förbättra arbetsprodukterna och sina färdigheter. För organisationen sparar defekter, som upptäcks och åtgärdas under testningen, tid och pengar och minskar de övergripande riskerna för produktkvaliteten.
- Informera om testresultat och andra resultat på ett neutralt och faktaorienterat sätt utan att kritisera personen som skapat det defekta objektet. Skriv objektiva felrapporter och granskningsresultat som bygger på fakta.
- Försök förstå hur den andra personen känner och orsakerna till varför personen kanske reagerar negativ på informationen.
- Kontrollera att den andra personen har förstått vad som sagts och tvärt om.

Typiska testmål har diskuterats tidigare (se avsnitt 1.1). Att tydligt definiera rätt uppsättning testmål har viktiga psykologiska konsekvenser. De flesta brukar anpassa sina planer och beteenden efter de mål som satts upp av teamet, ledningen och andra intressenter. Det är också viktigt att testarna följer dessa målsättningar med minimal personlig partiskhet.

1.5.2 Testarnas och utvecklarnas tankesätt

Utvecklare och testare tänker ofta på olika sätt. Det främsta målet med utveckling är att designa och bygga en produkt. Som vi diskuterat tidigare inkluderar målen för testningen att verifiera och validera produkten, hitta defekter innan release och så vidare. Detta är två olika målsättningar som kräver olika tankesätt. Genom att samla dessa båda tankesätt kan en högre produktkvalitet uppnås.

Ett tankesätt återspeglar en persons antaganden och prioriterade metoder för beslutsfattande och problemlösning. Testarens tankesätt bör omfatta nyfikenhet, yrkesmässig pessimism, kritisk iakttagelseförmåga, noggrannhet samt motivation för god och positiv kommunikation och goda relationer. Testarens tankesätt utvecklas och mognar ofta allteftersom testaren blir mer erfaren.

Utvecklarens tankesätt kan innehålla delar av testarens tankesätt, men framgångsrika utvecklare är ofta mer intresserade av att designa och bygga lösningar än att fundera på vad som kan vara fel med dessa lösningar. Dessutom kan bekräftelseförväntningar göra det svårt att hitta fel i det egna arbetet.

Med rätt tankesätt kan utvecklarna testa sin egen kod. Olika modeller för programvaruutvecklingens livscykel har ofta olika sätt att organisera testarna och testaktiviteterna. Genom att låta oberoende testare sköta en del av testaktiviteterna kan defekterna upptäckas mer effektivt, vilket är särskilt viktigt för stora, komplexa eller säkerhetskritiska system. Oberoende testare tillför ett perspektiv som skiljer sig från det som upphovsmännen till arbetsprodukten har (det vill säga affärsanalytiker, produktägare, designers och programmerare), eftersom de har andra kognitiva förutfattade meningar än upphovsmännen.

2 Testning under programvarans utvecklingslivscykel — 100 minuter

Nyckelord

acceptanstestning, acceptanstestning av förordningar, acceptanstestning av kontrakt, alfatestning, användaracceptanstestning, betatestning, driftacceptanstestning, funktionstestning, kommersiellt från hyllan (COTS), integrationstestning, komponentintegrationstestning, komponenttestning, omtestning, påverkansanalys, regressionstestning, testning av icke-funktionella egenskaper, sekventiell utvecklingsmodell, systemintegrationstestning, systemtestning, testbas, testfall, testmiljö, testmål, testnivå, testobjekt, testtyp, underhållstestning, white-box-testning

Utbildningsmål för testning under programvarans utvecklingslivscykel

2.1 Modeller för programvarans utvecklingslivscykel

FL-2.1.1 (K2) Förklara relationerna mellan utvecklingsaktiviteter för programvara och testaktiviteter i programvarans utvecklingslivscykel

FL-2.1.2 (K1) Identifiera orsakerna till att modeller för programvarans utvecklingslivscykel måste anpassas till projektets sammanhang och produkternas egenskaper

2.2 Testnivåer

FL-2.2.1 (K2) Jämföra de olika testnivåerna avseende målsättningar, testbas, testobjekt, typiska defekter och felsymptom samt arbetssätt och ansvarsområden

2.3 Testtyper

FL-2.3.1 (K2) Jämföra funktionell testning, icke-funktionell testning och white-box-testning

FL-2.3.2 (K1) Förstå att funktionell testning, icke-funktionell testning och white-box-testning finns på alla testnivåer

FL-2.3.3 (K2) Jämföra syftena med omtestning och regressionstestning

2.4 Underhållstestning

FL-2.4.1 (K2) Sammanfatta utlösande orsaker för underhållstestning

FL-2.4.2 (K2) Beskriva vilken funktion påverkansanalys har i underhållstestning

2.1 Modeller för programvarans utvecklingslivscykel

En modell för programvarans utvecklingslivscykel beskriver vilken typ av aktiviteter som utförs vid varje skede av ett utvecklingsprojekt för programvara, och hur dessa aktiviteter hänger samman logiskt och kronologiskt. Det finns ett antal olika utvecklingslivscykelmodeller för programvara som var och en kräver olika angreppssätt för testning.

2.1.1 Programvaruutveckling och testning av programvara

En viktig del av testarens roll är att känna till vanliga modeller för programvarans utvecklingslivscykel så att lämpliga testaktiviteter kan genomföras.

I alla modeller för programvarans utvecklingslivscykel finns flera kännetecken på bra testning:

- För varje utvecklingsaktivitet finns en motsvarande testaktivitet
- Varje testnivå har testmål som är specifika för den nivån
- Testanalys och testdesign för en viss testnivå påbörjas under motsvarande utvecklingsaktivitet
- Testarna deltar i diskussioner för att definiera och förfina krav och design, och deltar i granskningen av arbetsprodukterna (till exempel krav, design, användarberättelser etc.) så snart utkast finns tillgängliga

Oavsett vilken modell som väljs för programvarans utvecklingslivscykel, ska testaktiviteterna påbörjas tidigt under livscykeln enligt testprincipen för tidig testning.

Den här kursplanen klassificerar vanliga modeller för programvarans utvecklingslivscykel på följande sätt:

- Sekventiella utvecklingsmodeller
- Modeller för inkrementell och iterativ utveckling

En sekventiell utvecklingsmodell beskriver programvarans utvecklingsprocess som ett linjärt och sekventiellt flöde av aktiviteter. Detta betyder att alla faser i utvecklingsprocessen ska inledas när den föregående fasen är slutförd. Teoretiskt sett är det ingen överlappning mellan faserna, men i praktiken är det gynnsamt att få tidig återkoppling från efterföljande faser.

I vattenfallsmodellen slutförs utvecklingsaktiviteterna (till exempel kravanalyser, design, kodning och testning) efter varandra. I den modellen genomförs testaktiviteterna först när alla andra utvecklingsaktiviteter är slutförda.

Till skillnad från vattenfallsmodellen integrerar V-modellen testprocessen med utvecklingsprocessen och tillämpar principerna om tidig test. Dessutom inkluderar V-modellen de testnivåer som förknippas med motsvarande utvecklingsfas, vilket ytterligare stödjer den tidiga testningen (se avsnitt 2.2 för en diskussion om testnivåer). I den här modellen genomförs tester som associeras med varje testnivå sekventiellt, men i vissa fall förekommer överlappning.

Sekventiella utvecklingsmodeller levererar programvara som innehåller samtliga funktioner, men kräver normalt sett månader eller år för leverans till intressenter och användare.

Inkrementell utveckling omfattar krav, design, skapande och testning av systemet i delar, vilket betyder att programvarans funktioner växer inkrementellt. Storleken på inkrementen för funktionerna varierar, så att vissa metoder har större delar och andra har mindre. Funktionsinkrementen kan vara små som en enstaka förändring av en vy i användargränssnittet eller ett nytt frågealternativ.

Iterativ utveckling är när grupper av funktioner specificeras, designas, skapas och testas tillsammans i en serie med cykler, ofta med fast varaktighet. Iterationerna kan omfatta förändringar av funktioner som

utvecklats i tidigare iterationer, tillsammans med förändringar av projektets omfattning. Varje iteration levererar fungerande programvara som är en växande deluppsättning av den totala funktionsuppsättningen tills den slutliga programvaran levereras eller utvecklingen stoppas.

Här är några exempel:

- Rational Unified Process (RUP): Varje iteration tenderar att vara relativt lång (till exempel två till tre månader) och funktionsinkrementen är motsvarande stora, till exempel två eller tre grupper med relaterade funktioner
- Scrum: Varje iteration brukar vara relativt kort (till exempel timmar, dagar eller ett par veckor) och funktionsinkrementen är motsvarande små, till exempel ett par förbättringar och/eller två eller tre nya funktioner
- Kanban: Implementeras med eller utan iterationer av fast längd som kan leverera antingen en enda förbättring eller funktion vid färdigställande, eller gruppera funktioner så att de kan levereras i samtidig release
- Spiral (eller prototyputveckling): Omfattar att skapa experimentella inkrement, där en del kan omarbetas i stor omfattning eller till och med överges senare

Komponenter eller system som utvecklas med de här metoderna innefattar ofta överlappande och iterativa testnivåer under utvecklingen. Helst bör varje funktion testas på flera testnivåer på vägen mot leverans. I vissa fall använder teamen kontinuerliga leveranser (continuous delivery) eller kontinuerlig distribution (continuous deployment), som båda innebär betydande automatisering på flera testnivåer som del av leveransvägarna. Många arbetssätt för utveckling som använder dessa metoder inkluderar också konceptet med självorganiserande team, som kan ändra hur testarbetet organiseras samt relationen mellan testare och utvecklare.

Dessa metoder utgör ett snabbväxande system som kan levereras till slutanvändarna, funktion efter funktion, iteration efter iteration eller i mer traditionella större releaser. Oavsett om programvarans inkrement levereras till slutanvändarna, blir regressionstestningen allt viktigare i takt med att systemet växer.

Till skillnad från sekventiella modeller kan iterativa och inkrementella modeller leverera användbar programvara på veckor eller till och med dagar, men levererar endast den kompletta uppsättningen kravställda produkter under en period av månader eller till och med år.

Mer information om programvarutestning inom agil utveckling finns i ISTQB-AT Foundation Level Agile Tester Extension Syllabus, Black 2017, Crispin 2008 och Gregory 2015.

2.1.2 Programvarans utvecklingslivscykelmodell i ett sammanhang

Programvarans utvecklingscykelmodell måste väljas och anpassas till projektets sammanhang och produktegenskaperna. En lämplig modell för programvarans utvecklingslivscykel bör väljas och anpassas baserat på projektets mål, vilken typ av projekt som utvecklas, affärsprioriteringar (till exempel leveranstider) samt identifierade produkt- och projektrisker. Till exempel bör testning och utveckling av ett mindre internt administrativt system skilja sig från utveckling och testning av ett funktionssäkerhetskritiskt system, till exempel styrsystem för bilbromsar. Ett annat exempel kan i vissa fall vara att organisatoriska och kulturella problem försvårar kommunikationen mellan teammedlemmarna, vilket kan hindra den iterativa utvecklingen.

Beroende på projektets sammanhang kan det vara nödvändigt att kombinera eller omorganisera testnivåer och/eller testaktiviteter. Vid exempelvis integration av en programvaruprodukt som är kommersiell från hyllan (COTS) i ett större system, kan köparen köra interoperabilitetstestning på testnivån för systemintegration (till exempel integration med infrastrukturen och andra system) och på

acceptanstestnivå (funktionell och icke-funktionell, tillsammans med användaracceptanstestning och driftacceptanstestning). I avsnitt 2.2 finns en diskussion om testnivåer och i avsnitt 2.3 en diskussion om testtyper.

Dessutom kan själva modellerna för programvarans utvecklingslivscykel kombineras. V-modellen kan till exempel användas för att utveckla och testa backend-system och deras integration, medan en modell för agil utveckling kan användas för att utveckla och testa frontend-användargränssnitt (UI) och funktioner. Prototyper kan användas tidigt i projektet, medan en modell för inkrementell utveckling införs så snart experimentfasen är färdig.

System för sakernas internet (IoT) som består av många olika objekt, till exempel enheter, produkter och tjänster, använder normalt sett separata modeller för programvarans utvecklingslivscykel för varje objekt. Detta innebär en särskild utmaning för utvecklingen av versioner för system för sakernas internet. Dessutom lägger programvarans utvecklingslivscykel för sådana objekt större vikt vid de senare faserna efter att objekten har tagits i drift (till exempel driftfas, uppdateringsfas och avvecklingsfas).

2.2 Testnivåer

Testnivåer är grupper av testaktiviteter som organiseras och hanteras tillsammans. Varje testnivå är en instans av testprocessen som består av de aktiviteter som beskrivs i avsnitt 1.4 och utförs i relation till programvaran på en given utvecklingsnivå, från enskilda enheter och komponenter till kompletta system eller (i förekommande fall) system av system. Testnivåerna är relaterade till andra aktiviteter inom programvarans utvecklingslivscykel. De testnivåer som används i den här kursplanen är:

- Komponenttestning
- Integrationstestning
- Systemtestning
- Acceptanstestning

Testnivåerna kännetecknas av följande attribut:

- Specifika mål
- Testbas, hänvisad för att härleda testfall
- Testobjekt (det vill säga det som testas)
- Typiska defekter och felsymptom
- Specifika arbetssätt och ansvarsområden

För varje testnivå krävs en lämplig testmiljö. Under acceptanstestning är till exempel en produktionsliknande testmiljö idealisk, medan utvecklarna vid komponenttestning normalt sett använder sin egen utvecklingsmiljö.

2.2.1 Komponenttestning

Mål med komponenttestningen

Komponenttestning (även kallat enhets- eller modultestning) fokuserar på komponenter som går att testa separat. Målen med komponenttestningen omfattar:

- Riskminskning

- Verifiera att de funktionella eller icke-funktionella beteendena hos komponenten fungerar enligt design och specifikation
- Skapa förtroende för komponentens kvalitet
- Hitta defekter hos komponenten
- Förhindra att defekter släpps igenom till högre testnivåer

I vissa fall, särskilt vid inkrementella och iterativa utvecklingsmodeller (till exempel agil utveckling) och kontinuerliga kodförändringar, spelar automatiserade regressionstester för komponenterna en viktig roll när det gäller att skapa förtroende för att ändringarna inte skadat befintliga komponenter.

Komponenttestning genomförs ofta isolerat från resten av systemet beroende på programvarans utvecklingslivscykelmodell och systemet, som kan kräva falska objekt, tjänstevirtualisering, exekveringsplattformar, stubbar och drivrutiner. Komponenttestning kan täcka funktionella egenskaper (till exempel om beräkningar är korrekta), icke-funktionella egenskaper (till exempel att söka efter minnesläckor) och strukturella egenskaper (till exempel beslutstestning).

Testbas

Exempel på arbetsprodukter som kan användas som testbas vid komponenttestning:

- Detaljerad design
- Kod
- Datamodell
- Komponentspecifikationer

Testobjekt

Typiska testobjekt för komponenttestning:

- Komponenter, enheter eller moduler
- Kod- och datastrukturer
- Klasser
- Databasmoduler

Typiska defekter och felsymptom

Exempel på typiska defekter och felsymptom vid komponenttestning:

- Felaktiga funktioner (till exempel inte enligt beskrivningen i designspecifikationerna)
- Dataflödesproblem
- Felaktig kod och logik

Defekter åtgärdas vanligtvis så snart de upptäcks, oftast utan formell felhantering. Men när utvecklare rapporterar defekterna ger detta viktig information för analys av grundorsaker och processförbättringar.

Specifika arbetssätt och ansvarsområden

Komponenttestning genomförs vanligtvis av den utvecklare som skrev koden, men kräver åtminstone åtkomst till koden som testas. Utvecklarna kan växla mellan att utveckla komponenter och att hitta och åtgärda fel. Utvecklarna skriver och exekverar ofta tester efter att ha skrivit koden för en komponent.

Under framför allt agil utveckling kan dock skrivande av automatiserade testfall för komponenter föregå skrivandet av applikationskod.

Ett exempel är testdriven utveckling (TDD). Testdriven utveckling är mycket iterativ och bygger på cykler med utveckling av automatiserade testfall. Dessa följs av att bygga och integrera små kodstycken och därefter köra komponenttesterna, korrigera eventuella problem och sedan omstrukturera koden. Processen fortsätter tills komponenten är helt färdigbyggd och alla komponenttester är godkända. Testdriven utveckling är ett exempel på ett ”test first”-arbetssätt. Även om testdriven utveckling har sitt ursprung i extrem programmering (XP), har den spridit sig till andra former av agil testning och sekventiella livscykler (se ISTQB-AT Foundation Level Agile Tester Extension Syllabus).

2.2.2 Integrationstestning

Mål med integrationstestningen

Integrationstestning fokuserar på interaktioner mellan komponenter eller system. Målen med integrationstestningen omfattar:

- Riskminskning
- Verifiera att de funktionella eller icke-funktionella beteendena hos gränssnitten fungerar enligt design och specifikation
- Skapa förtroende för gränssnittens kvalitet
- Hitta defekter (som kan finnas i själva gränssnitten eller i komponenterna och systemen)
- Förhindra att defekter släpps igenom till högre testnivåer

Precis som vid komponenttestning kan automatiserad regressionstestning vid integration i vissa fall skapa förtroende för att ändringarna inte har skadat befintliga gränssnitt, komponenter eller system.

Två olika nivåer av integrationstestning beskrivs i den här kursplanen och kan utföras på testobjekt av varierande storlek enligt följande:

- Komponentintegrationstestning fokuserar på interaktioner och gränssnitt mellan integrerade komponenter. Komponentintegrationstestning utförs efter komponenttestningen och är i allmänhet automatiserad. Under iterativ och inkrementell utveckling sker komponentintegrationstesterna vanligtvis som del av den kontinuerliga integrationsprocessen.
- Systemintegrationstestning fokuserar på interaktioner och gränssnitt mellan system, paket och mikrotjänster. Systemintegrationstestning kan också omfatta interaktioner med och gränssnitt som tillhandahålls av externa organisationer (till exempel webbtjänster). I det här fallet styr inte den utvecklande organisationen de externa gränssnitten, vilket kan skapa olika utmaningar för testningen. Det kan till exempel vara att säkerställa att testblockerande defekter i den externa organisationens kod blir lösta eller att konfigurera testmiljöerna. Systemintegrationstest kan utföras efter systemtest eller parallellt med pågående systemtestaktiviteter (i både sekventiell utveckling samt iterativ och inkrementell utveckling).

Testbas

Exempel på arbetsprodukter som kan användas som testbas vid integrationstestning:

- Design av system och programvara
- Sekvensdiagram
- Specifikationer för gränssnitts och kommunikationsprotokoll

- Användningsfall
- Arkitektur på komponent- eller systemnivå
- Arbetsflöden
- Externa gränssnittsdefinitioner

Testobjekt

Typiska testobjekt för integrationstestning:

- Delsystem
- Databaser
- Infrastruktur
- Gränssnitt
- API:er
- Mikrotjänster

Typiska defekter och felsymptom

Exempel på typiska defekter och felsymptom vid komponentintegrationstestning:

- Felaktiga data, saknade data eller felaktig datakodning
- Felaktig ordningsföljd eller tid för gränssnittsanrop
- Olikheter i gränssnitten
- Felsymptom i kommunikationen mellan komponenter
- Ohanterade eller felaktigt hanterade kommunikationsfel mellan komponenter
- Felaktiga antaganden kring innebörd, enheter eller gränser för de data som skickas mellan komponenter

Exempel på typiska defekter och felsymptom vid systemintegrationstestning:

- Inkonsekventa meddelandestrukturer mellan system
- Felaktiga data, saknade data eller felaktig datakodning
- Olikheter i gränssnitten
- Felsymptom i kommunikationen mellan system
- Ohanterade eller felaktigt hanterade kommunikationsfel mellan system
- Felaktiga antaganden kring innebörd, enheter eller gränser för de data som skickas mellan system
- Underlåtenhet att uppfylla obligatoriska säkerhetskrav

Specifika arbetssätt och ansvarsområden

Komponentintegrationstestning och systemintegrationstestning bör fokusera på själva integrationen. Till exempel vid integration av modul A med modul B bör testerna fokusera på kommunikationen mellan modulerna, inte funktionerna hos enskilda moduler, eftersom detta bör ha gjorts under komponenttestningen. Vid integration av system X med system Y bör testerna fokusera på

kommunikationen mellan systemen, inte funktionerna hos enskilda system, eftersom detta bör ha gjorts under systemtestningen. Funktionella, icke-funktionella och strukturella testtyper kan användas.

Det är ofta utvecklarna som ansvarar för komponentintegrationstesterna. Det är i allmänhet testarna som ansvarar för systemintegrationstesterna. Helst bör testarna som utför systemintegrationstester förstå systemarkitekturen och dessutom bör de ha påverkat planeringen av integrationen.

Om integrationstesterna och integrationsstrategin planeras innan komponenterna eller systemen byggs, kan dessa komponenter eller system byggas i den ordning som krävs för effektivast möjliga testning. Systematiska integrationsstrategier kan bygga på systemarkitekturen (till exempel uppifrån-ned och nedifrån-upp), funktionella uppgifter, sekvenser för transaktionsbearbetning eller andra aspekter av systemet eller komponenterna. För att förenkla tidig upptäckt och isolering av defekter bör integrationen normalt sett vara inkrementell (det vill säga ett litet antal tillagda komponenter eller system i taget) snarare än en ”big bang” (det vill säga att integrera alla komponenter eller system i ett enda steg). En riskanalys av de mest komplexa gränssnitten kan bidra till att fokusera integrationstestningen.

Ju större omfattningen av integrationen, desto svårare blir det att isolera defekter till en specifik komponent eller ett specifikt system, vilket kan leda till ökad risk och att felsökningen tar längre tid. Det här är en anledning till att kontinuerlig integration där programvaran integreras en komponent i taget (det vill säga funktionell integration) har blivit allt vanligare. Sådan kontinuerlig integration inkluderar ofta automatisk regressionstestning, helst på flera testnivåer.

2.2.3 Systemtestning

Mål med systemtestningen

Systemtestning fokuserar på beteenden och kapacitet hos ett helt system eller en komplett produkt, ofta med hänsyn till de end-to-end-uppgifter som systemet kan utföra och de icke-funktionella beteenden systemet visar upp medan uppgifterna utförs. Målen med systemtestningen:

- Riskminskning
- Verifiera att de funktionella eller icke-funktionella beteendena hos systemen fungerar enligt design och specifikation
- Validera att systemet är komplett och fungerar som förväntat
- Skapa förtroende för kvaliteten hos systemet som helhet
- Hitta defekter
- Förhindra att defekter släpps igenom till högre testnivåer eller i produktionen

För vissa system kan verifiering av datakvaliteten vara ett mål. Precis som vid komponenttestning och integrationstestning kan automatiserad systemregressionstestning i vissa fall skapa förtroende för att ändringarna inte har skadat befintliga funktioner eller end-to-end-funktionaliteten. Systemtestningen producerar ofta information som används av intressenter för att fatta beslut om release. Systemtestningen kan också uppfylla legala eller reglerande krav eller standarder.

Testmiljön bör helst motsvara slutmålet eller produktionsmiljön.

Testbas

Exempel på arbetsprodukter som kan användas som testbas vid systemtestning omfattar:

- Kravspecifikationer för system och programvara (funktionella och icke-funktionella)
- Riskanalysrapporter

- Användningsfall
- Epics och användarberättelser
- Modeller av systemets beteende
- Tillståndsdiagram
- System- och användarmanualer

Testobjekt

Typiska testobjekt för systemtestning:

- Applikationer
- Hårdvaru-/programvarusystem
- Operativsystem
- Systemet som testas (SUT)
- Systemkonfiguration och konfigurationsdata

Typiska defekter och felsymptom

Exempel på typiska defekter och felsymptom vid systemtestning omfattar:

- Felaktiga beräkningar
- Felaktigt eller oväntat funktionellt eller icke-funktionellt beteende för systemet
- Felaktig styrning och/eller felaktiga dataflöden i systemet
- Det går inte att genomföra end-to-end-funktionstestning på ett korrekt och komplett sätt
- Systemet fungerar inte korrekt i en eller flera produktionsmiljöer
- Systemet fungerar inte enligt beskrivningen i system- och användarmanualerna

Specifika angreppssätt och ansvarsområden

Systemtestningen bör fokusera på det övergripande end-to-end-beteendet hos systemet som helhet, både funktionellt och icke-funktionellt. Systemtestningen ska använda de mest lämpliga teknikerna (se kapitel 4) för aspekterna hos systemet som ska testas. En beslutstabell kan till exempel skapas för att verifiera att det funktionella beteendet är enligt beskrivningen i verksamhetsreglerna.

Systemtestningen utförs normalt sett av oberoende testare. Defekter i specifikationerna (till exempel saknade användarberättelser och felaktigt formulerade verksamhetskrav) kan leda till bristande förståelse för, eller oenighet kring, det förväntade systembeteendet. Sådana situationer kan orsaka falska positiva eller falska negativa resultat, vilket kan ge tidsförluster respektive minska effektiviteten i identifieringen av defekten. Att göra testarna delaktiga i förfiningen av användarberättelsen eller de statiska testaktiviteterna, till exempel granskningar, bidrar till att minska förekomsten av sådana situationer.

2.2.4 Acceptanstestning

Mål med acceptanstestningen

Acceptanstestning fokuserar liksom systemtestning normalt sett på beteenden och funktioner för ett helt system eller en hel produkt. Målen med acceptanstestningen omfattar:

- Etablera förtroende för kvaliteten hos systemet som helhet

- Validera att systemet är komplett och fungerar som förväntat
- Verifiera att de funktionella eller icke-funktionella beteendena hos systemet fungerar enligt specifikation

Acceptanstestning kan ta fram information för att utvärdera hur redo systemet är för införande och användning av kunden (slutanvändaren). Defekter kan upptäckas under acceptanstestning, men målet är oftast inte att hitta defekter. Att hitta ett betydande antal defekter under acceptanstestningen kan i vissa fall betraktas som en stor projektrisk. Acceptanstestningen kan också uppfylla legala eller reglerande krav eller standarder.

Vanliga former av acceptanstestning omfattar:

- Användaracceptanstestning
- Driftacceptanstestning
- Acceptanstestning av kontrakt och förordningar
- Alfa- och betatestning

Var och en beskrivs i följande fyra underavsnitt.

Användaracceptanstestning

Användarnas acceptanstestning av systemet fokuserar normalt sett på att validera systemets lämplighet för användning av de avsedda användarna i en verklig eller simulerad driftsmiljö. Huvudmålet är att skapa förtroende för att användarna kan använda systemet för att uppfylla sina behov, uppfylla gällande krav och utföra verksamhetsprocesser med minsta möjliga svårighet, kostnad och risk.

Driftacceptanstestning

Acceptanstestningen av systemet genomförs vanligtvis av verksamhets- eller systemadministrationspersonalen i en (simulerad) verksamhetsmiljö. Testerna fokuserar på verksamhetsaspekter och kan omfatta:

- Testning av säkerhetskopiering och återställning
- Installation, avinstallation och uppgradering
- Haveriberedskap
- Användarhantering
- Underhållsuppgifter
- Dataladdning och migrationsåtgärder
- Kontroller av säkerhetssårbarheter
- Prestandatestning

Huvudmålet med driftacceptanstestning är att skapa förtroende för att operatörerna eller systemadministratörerna kan få systemet att fungera korrekt för användarna i driftsmiljön, även under exceptionella eller svåra förhållanden.

Acceptanstestning av kontrakt och förordningar

Acceptanstestning av kontrakt genomförs mot ett kontrakts acceptanskriterier för att producera specialutvecklad programvara. Acceptanskriterier ska definieras när parterna kommer överens om kontraktet. Acceptanstestning av kontrakt utförs ofta av användare eller oberoende testare.

Acceptanstester av förordningar utförs mot alla förordningar som måste följas, till exempel myndighetsförordningar, legala förordningar eller säkerhetsförordningar. Acceptanstestning av förordningar utförs ofta av användare eller oberoende testare, och ibland bevittnas eller granskas resultaten av olika tillsynsmyndigheter.

Huvudmålet för acceptanstestning av kontrakt och förordningar är att skapa förtroende för att kontrakt och förordningar är uppfyllda.

Alfa- och betatestning

Alfa- och betatestning används normalt sett av utvecklare av programvara som är kommersiell från hyllan (COTS) och vill ha återkoppling från potentiella eller befintliga användare, kunder och/eller operatörer innan programvaruprodukten släpps på marknaden. Alfatestning utförs hos utvecklingsföretaget, men inte av utvecklingsteamet utan av potentiella eller befintliga kunder och/eller operatörer eller ett oberoende testteam. Betatestning utförs av potentiella eller befintliga kunder och/eller operatörer i sina egna miljöer. Betatestning kan komma efter alfatestningen, eller så kan den utföras utan att någon föregående alfatestning har genomförts.

Ett mål med alfa- eller betatestningen är att skapa förtroende bland potentiella eller befintliga kunder och/eller operatörer för att de kan använda systemet under normala vardagliga förhållanden och driftsmiljöer så att de kan uppnå sina mål med minsta möjliga svårighet, kostnad eller risk. Ett annat mål kan vara att upptäcka defekter som rör de villkor och miljöer där systemet ska användas, särskilt när dessa villkor eller miljöer är svåra för utvecklingsteamet att återskapa.

Testbas

Exempel på arbetsprodukter som kan användas som testbas vid alla former av acceptanstestning omfattar:

- Verksamhetsprocesser
- Användar- eller verksamhetskrav
- Regelverk, legala kontrakt och standarder
- Användningsfall
- Systemkrav
- System- eller användardokumentation
- Installationsprocesser
- Riskanalysrapporter

Dessutom kan en eller flera arbetsprodukter användas som testbas för att härleda testfall för driftacceptanstestning:

- Procedurer för säkerhetskopiering och återställning
- Procedurer för haveriberedskap
- Icke-funktionella krav
- Driftsdokumentation
- Implementerings- och installationshandledningar
- Prestandamål
- Databaspaket

- Säkerhetsstandarder och regelverk

Typiska testobjekt

Typiska testobjekt för alla slags acceptanstestning omfattar:

- Systemet som testas
- Systemkonfiguration och konfigurationsdata
- Verksamhetsprocesser för ett helt integrerat system
- Återhämtningssystem och ”hot sites” (för kontinuitet i verksamheten och testning av haveriberedskap)
- Drifts- och underhållsprocesser
- Formulär
- Rapporter
- Befintliga och konverterade produktionsdata

Typiska defekter och felsymptom

Exempel på typiska defekter vid alla former av acceptanstestning omfattar:

- Systemarbetsflödena uppfyller inte verksamhets- eller användarkraven
- Verksamhetsrelaterade regler har inte implementerats korrekt
- Systemet uppfyller inte kontraktsmässiga eller reglerande krav
- Icke-funktionella felsymptom som säkerhetssårbarheter, bristande prestandauppfyllelse vid hög belastning eller felaktig drift på en plattform som stöds

Specifika angreppssätt och ansvarsområden

Acceptanstestning är ofta kundernas, verksamhetsanvändarnas, produktanvändarnas eller systemoperatörernas ansvar, men även andra intressenter kan vara inblandade.

Acceptanstestning betraktas ofta som den sista testnivån i en sekventiell utvecklingscykel, men den kan också inträffa vid andra tidpunkter, till exempel:

- Acceptanstestning av programvaruprodukter som är kommersiella från hyllan (COTS) kan inträffa när de installeras eller integreras
- Acceptanstestning av en ny funktionell förbättring kan göras innan systemtestningen

Vid iterativ utveckling kan projektteamen använda olika former av acceptanstestning under och i slutet av varje iteration, till exempel sådana som är fokuserade på att verifiera en ny funktion mot acceptanskriterierna och sådana som fokuserar på att validera att en ny funktion uppfyller användarens behov. Dessutom kan alfatestning och betatestning förekomma antingen i slutet av varje iteration, efter att varje iteration har slutförts eller efter att en serie iterationer har slutförts. Användaracceptanstestning, driftsacceptanstestning och acceptanstestning av kontrakt och förordningar kan också förekomma antingen i slutet av varje iteration, efter att varje iteration har slutförts eller efter att en serie iterationer har slutförts.

2.3 Testtyper

En testtyp är en grupp av testaktiviteter som syftar till att testa specifika egenskaper för ett programvarusystem, eller en del av ett system, baserat på specifika testmål. Sådana mål kan omfatta:

- Utvärdering av funktionella kvalitetsegenskaper, till exempel fullständighet, korrekthet och lämplighet
- Utvärdering av icke-funktionella kvalitetsegenskaper, till exempel tillförlitlighet, prestandauppfyllelse, informationssäkerhet, kompatibilitet och användbarhet
- Utvärdering av om strukturen eller arkitekturen för komponenten eller systemet är korrekt, komplett och enligt specifikationerna
- Utvärdering av effekterna av ändringar, till exempel att bekräfta att defekterna har åtgärdats (omtestning) och att leta efter oavsedda beteendeändringar som uppstått efter ändringar av programvara eller miljö (regressionstestning)

2.3.1 Funktionell testning

Funktionell testning av ett system omfattar tester som utvärderar de funktioner som systemet bör utföra. Funktionella krav kan beskrivas i arbetsprodukterna som specifikationer för verksamhetskrav, epics, användarberättelser, användningsfall eller funktionsspecifikationer, eller så kan de vara odokumenterade. Funktionerna är ”vad” systemet bör göra.

Funktionella tester ska genomföras på alla testnivåer (till exempel för komponenter som kan bygga på en komponentspecifikation), även om fokus är olika på varje nivå (se avsnitt 2.2).

Funktionella tester handlar om programvarans beteende, så black-box-tekniker kan användas för att härleda testvillkor och testfall för funktionerna i komponenten eller systemet (se avsnitt 4.2).

Grundligheten i funktionell testning kan mätas genom täckning av funktionaliteten. Täckningsgraden av funktionaliteten är den omfattning i vilken någon typ av funktionselement har utövats av testerna, och uttrycks som en procent av de typer av element som täcks. Genom att till exempel använda spårbarhet mellan tester och funktionella krav, kan procenten av dessa krav som åtgärdats genom testningen beräknas för att eventuellt identifiera luckor i täckningsgraden.

Design och exekvering av funktionella tester kan innefatta särskilda färdigheter eller kunskaper, till exempel kunskap om det specifika verksamhetsproblem som programvaran löser (exempelvis programvara för geografisk modellering för olje- och gasindustrin) eller den specifika roll som programvaran har (exempelvis programvara för datorspel som ger interaktiv underhållning).

2.3.2 Icke-funktionell testning

Icke-funktionell testning av ett system utvärderar egenskaper hos system och programvara, till exempel användbarhet, prestandauppfyllelse eller informationssäkerhet. Se ISO-standard (ISO/IEC 25010) för en klassificering av programvaruproduktens kvalitetsegenskaper. Icke-funktionell testning är testning av ”hur väl” systemet beter sig.

Tvärt emot en vanlig missuppfattning kan och ofta ska, icke-funktionell testning utföras på alla nivåer och genomföras så tidigt som möjligt. En sen upptäckt av icke-funktionella defekter kan utgöra extremt stora risker för ett lyckat projekt.

Black-box-tekniker (se avsnitt 4.2) kan användas för att härleda testvillkor och testfall för icke-funktionell testning. Gränsvärdesanalys kan till exempel användas för att definiera stressvillkoren för prestandatester.

Grundligheten i icke-funktionell testning kan mätas genom den icke-funktionella täckningsgraden. Den icke-funktionella täckningsgraden är den omfattning i vilken några typer av icke-funktionella element har utövats av testerna, och uttrycks som ett procenttal av de typer av element som täcks. Genom att till exempel använda spårbarhet mellan tester och enheter för en mobilapplikation, kan procenten av de enheter som hanterats med kompatibilitetstestningen beräknas för att eventuellt identifiera luckor i täckningsgraden.

Design och exekvering av icke-funktionella tester kan innefatta särskilda färdigheter eller kunskaper, till exempel kunskap om den inneboende svagheten i en design eller teknik (till exempel säkerhetssårbarheter förknippade med vissa programspråk) eller den specifika användarbasen (till exempel användarna av olika ledningssystem vid en vårdinrättning).

I ISTQB-ATA Advanced Level Test Analyst Syllabus, ISTQB-TTA Advanced Level Technical Test Analyst Syllabus, ISTQB-SEC Advanced Level Security Tester Syllabus och andra ISTQB-specialistmoduler finns mer information om egenskaperna för icke-funktionell testning.

2.3.3 White-box-testning

White-box-testning härleder sina tester baserat på systemets interna struktur eller implementering. Intern struktur kan inkludera kod, arkitektur, arbetsflöden och/eller dataflöden i systemet (se avsnitt 4.3).

Grundligheten i white-box-testningen kan mätas genom strukturtäckning. Strukturtäckning är den omfattning i vilken någon typ av strukturelement har exekverats av testerna, och uttrycks som en procent av de typer av element som täcks.

På komponenttestnivå är kodtäckningen baserad på procenten av komponentkoden som har testats, och kan mätas som olika aspekter av koden (täckningsobjekt), till exempel procenten av exekverbara satser som har testats i komponenten eller procenten av beslutsutfall som har testats. Dessa typer av täckningsgrad kallas gemensamt för kodtäckning. På komponentintegrationstestnivå kan white-box-testning bygga på systemets arkitektur, till exempel programgränssnitt mellan komponenterna, och strukturtäckning kan mätas i procent av de gränssnitt som exekverats av testerna.

Design och utförande av white-box-testning kan innefatta särskilda kunskaper eller färdigheter, till exempel om hur koden är byggd (genom att till exempel använda kodtäckningsverktyg), hur data lagras (genom att till exempel utvärdera möjliga databasfrågor), användning av verktyg för test av täckningsgrad samt att tolka resultaten på rätt sätt.

2.3.4 Ändringsrelaterad testning

När ett system ändras, antingen för att korrigera en defekt eller på grund av nya eller ändrade funktioner, ska testning genomföras för att bekräfta att ändringarna har korrigerat defekten eller att funktionen implementerats på rätt sätt, och inte har orsakat några oförutsedda negativa konsekvenser.

- Omtestning: När en defekt har åtgärdats kan programvaran testas med alla testfall som inte godkändes på grund av defekten, och bör sedan köras på nytt med den nya versionen av programvaran. Programvaran kan också testas med nya tester om defekten till exempel innebar saknade funktioner. Åtminstone ska stegen för att återskapa de felsymptom som orsakades av defekten köras på nytt för den nya versionen av programvaran. Syftet med omtestning är att bekräfta att den ursprungliga defekten har åtgärdats.
- Regressionstestning: Det är möjligt att en ändring som görs i en del av koden, oavsett om det gäller en korrigering eller en annan typ av ändring, oavsiktligt kan påverka beteendet hos andra delar av koden, antingen inom samma komponent, i andra komponenter som tillhör samma system eller till och med i andra system. Ändringar kan omfatta ändringar av miljön, till exempel en ny version av ett operativsystem eller ett databashanteringssystem. Sådana oavsiktliga

bieffekter kallas för regression. Regressionstestning innefattar exekvering av de tester som ska identifiera sådana oavsiktliga bieffekter.

Omtestning och regressionstestning genomförs på alla testnivåer.

Särskilt vid iterativa och inkrementella utvecklingslivscykler (till exempel agila) medför nya funktioner, ändringar av befintliga funktioner och omstrukturering av koden till ändringar av koden, vilket också kräver ändringsrelaterad testning. På grund av systemets föränderliga natur är omtestning och regressionstestning mycket viktigt. Det här är särskilt relevant för system med sakernas internet där enskilda objekt (till exempel enheter) ofta uppdateras eller byts ut.

Regressionstestsviter körs många gånger och förändras i allmänhet långsamt, vilket gör regressionstestning till en stark kandidat för automatisering. Automatisering av dessa tester bör inledas tidigt under projektet (se kapitel 6).

2.3.5 Testtyper och testnivåer

Det är möjligt att utföra alla ovanstående testtyper på alla testnivåer. Som en illustration ges här exempel på funktionella tester, icke-funktionella tester, white-box-tester och ändringsrelaterade tester på alla testnivåer för en bankapplikation. Följande är exempel på funktionella tester:

- För komponenttestning designas testerna baserat på hur en komponent bör beräkna sammansatt ränta.
- För komponentintegrationstestning designas testerna baserat på hur kontoinformation som samlas in via användargränssnittet förs vidare till affärslogiken.
- För systemtestning designas testerna baserat på hur kontoinnehavarna kan ansöka om kredit för sina checkkonton.
- För systemintegrationstestning designas testerna baserat på hur systemet använder en extern mikrotjänst för att kontrollera en kontoinnehavares kreditvärde.
- För acceptanstestning designas testerna baserat på hur banken hanterar godkännande eller avslag för kreditansökningar.

Följande är exempel på icke-funktionella tester:

- För komponenttestning designas prestandatesterna för att utvärdera antalet CPU-cykler som krävs för att utföra en komplex beräkning av totalräntan.
- För komponentintegrationstestning designas informationssäkerhetstesterna för säkerhetsrisker vid buffertöverflöde på grund av data som skickas till gränssnittet från affärslogiken.
- För systemtestning designas portabilitetstestningen för att kontrollera om presentationslagret fungerar på alla webbläsare och mobila enheter som stöds.
- För systemintegrationstestning designas tillförlitlighetstesterna för att utvärdera systemets stabilitet om mikrotjänsten för kreditvärdet inte svarar.
- För acceptanstestning designas användbarhetstesterna för att utvärdera tillgängligheten för bankens kreditbehandlingsgränssnitt för personer med funktionsnedsättningar.

Följande är exempel på white-box-tester:

- För komponenttestning designas testerna för att uppnå fullständig kodsats- och beslutstäckning (se avsnitt 4.3) för alla komponenter som utför ekonomiska beräkningar.

- För komponentintegrationstestning designas testerna för att bekräfta att varje skärm i webbläsarens gränssnitt överlämnar data till nästa skärm och till affärslogiken.
- För systemtestning designas testerna för att täcka sekvenser av webbsidor som kan uppstå under en kreditansökan.
- För systemintegrationstestning designas testerna för att exekvera alla möjliga frågetyper som skickas till mikrotjänsten för kreditvärde.
- För acceptanstestning designas testerna för att täcka alla datafilstrukturer och värdeintervall som stöds för överföringar mellan banker.

Slutligen följer några exempel på ändringsrelaterade tester:

- För komponenttestning byggs automatiserade regressionstester för varje komponent och inkluderas i ramverket för kontinuerlig integration.
- För komponentintegrationstestning designas testerna för att bekräfta gränssnittsrelaterade defekter när korrigeringarna checkas in i databasen för källkod.
- För systemtestning körs alla tester för ett angivet arbetsflöde igen om någon skärm i arbetsflödet ändras.
- För systemsintegrationstestning körs tester för applikationens dagliga interaktion med mikrotjänsten för kreditbedömning på nytt varje dag som en del av den kontinuerliga implementeringen av mikrotjänsten.
- För acceptanstestning körs alla tester, som tidigare inte har godkänts, på nytt efter att en defekt som hittats i acceptanstestningen har åtgärdats.

Även om det här avsnittet innehåller exempel på varje testtyp på alla nivåer, är det inte nödvändigt för all programvara att ha varje testtyp representerad på varje nivå. Det är dock viktigt att köra tillämpliga testtyper på varje nivå, särskilt på den tidigaste nivån där testtypen förekommer.

2.4 Underhållstestning

När programvara och system har implementerats i produktionsmiljöerna måste underhållet skötas. Ändringar av olika slag är nästan oundvikliga i levererad programvara och system, antingen för att åtgärda defekter under operativ användning, för att lägga till nya funktioner eller för att ta bort eller ändra funktioner som redan har levererats. Underhåll krävs också för att bibehålla eller förbättra icke-funktionella kvalitetsegenskaper för komponenten eller systemet över dess livslängd, särskilt prestandauppfyllelse, kompatibilitet, tillförlitlighet, informationssäkerhet och portabilitet.

När ändringar görs som del av underhållet bör underhållstestning genomföras, både för att utvärdera hur lyckade de genomförda ändringarna har varit och för att leta efter möjliga bieffekter (till exempel regression) i delar av systemet som förblir oförändrade (vilket vanligtvis är huvuddelen av systemet). Underhållstestning fokuserar på att testa ändringar av systemet samt att testa oförändrade delar som kan ha påverkats av ändringarna. Underhåll kan omfatta planerade och oplanerade releaser (snabbkorrigeringar, ”hot fix”).

En underhållsrelease kan kräva underhållstestning på flera testnivåer och med olika testtyper beroende på omfattningen. Underhållstestningens omfattning beror på:

- Risker med ändringen, till exempel i vilken grad det ändrade området i programvaran delar information med andra komponenter eller system
- Storleken på det befintliga systemet

- Storleken på ändringen

2.4.1 Utlösande faktorer för underhåll

Det finns flera orsaker till varför underhåll av programvara – och därmed underhållstestning – genomförs, både för planerade och oplanerade ändringar.

Vi kan klassificera utlösande faktorer för underhåll enligt följande:

- Modifiering, till exempel planerade förbättringar (exempelvis releasebaserade), korrigerande och brådskande ändringar, ändringar av driftsmiljön (exempelvis planerade uppgraderingar av operativsystemet eller databasen), uppgraderingar av programvara som är kommersiell från hyllan (COTS) och patchar för defekter och sårbarheter
- Migrering (till exempel från en plattform till en annan) som kan kräva driftstester av den nya miljön och den ändrade programvaran, eller tester av datakonvertering när data från en annan applikation ska migreras till systemet där underhållet sker
- Tillbakadragning, till exempel när en applikation når slutet av sin livslängd

När en applikation eller ett system dras tillbaka kan det kräva testning av datamigrering eller arkivering om data ska sparas under långa perioder. Testning av procedurer för återställning/hämtning efter långvarig arkivering kan också behövas. Dessutom kan regressionstestning behövas för att säkerställa att alla funktioner som fortfarande används fungerar.

För system med sakernas internet kan underhållstestning utlösas genom att införa helt nya eller modifierade saker i det övergripande systemet, till exempel hårdvaruenheter eller programvarutjänster. Underhållstestningen för sådana system lägger särskild vikt vid integrationstestning på olika nivåer (till exempel nätverksnivå, applikationsnivå) och säkerhetsaspekter, framför allt de som rör personuppgifter.

2.4.2 Påverkansanalys för underhåll

Påverkansanalyser utvärderar de förändringar som gjorts i en underhållsrelease i syfte att identifiera de avsedda konsekvenserna samt förväntade och möjliga bieffekter av en ändring, samt identifiera de områden i systemet som kommer att påverkas av förändringen. Påverkansanalyser kan också bidra till att identifiera vilken påverkan en ändring har på befintliga tester. Bieffekter och påverkade områden i systemet behöver testas för regression, möjligen efter att eventuella befintliga tester som påverkas av ändringen har uppdaterats.

Påverkansanalyser kan utföras innan en ändring genomförs för att hjälpa till att avgöra om ändringen bör göras, baserat på möjliga konsekvenser i andra områden av systemet.

Påverkansanalyser kan vara svåra att genomföra om:

- Specifikationerna (till exempel verksamhetskrav, användarberättelser och arkitektur) är för gamla eller saknas
- Testfallen är inte dokumenterade eller för gamla
- Dubbelriktad spårbarhet mellan tester och testbasen har inte bibehållits
- Verktygsstödet är bristfälligt eller obefintligt
- De inblandade personerna saknar kunskap om domänen och/eller systemet
- Otillräcklig uppmärksamhet har ägnats åt programvarans underhållbarhet under utvecklingen

3 Statisk testning — 135 minuter

Nyckelord

ad hoc-granskning, checklistebaserad granskning, dynamisk testning, formell granskning, genomgång, granskning, informell granskning, inspektion, perspektivbaserad läsning, rollbaserad granskning, scenariobaserad granskning, statisk analys, statisk testning, teknisk granskning

Utbildningsmål för statisk testning

3.1 Grunder för statisk testning

FL-3.1.1 (K1) Känna igen typer av arbetsprodukter för programvara som kan granskas med de olika statiska testteknikerna

FL-3.1.2 (K2) Använda exempel för att beskriva värdet av statisk testning

FL-3.1.3 (K2) Förklara skillnaden mellan statiska och dynamiska tekniker med hänsyn till mål, typer av defekter som ska identifieras och vilken roll dessa tekniker spelar inom programvarans livscykel

3.2 Granskningsprocess

FL-3.2.1 (K2) Sammanfatta aktiviteterna för arbetsproduktens granskningsprocess

FL-3.2.2 (K1) Känna igen de olika rollerna och ansvarsområdena i en formell granskning

FL-3.2.3 (K2) Förklara skillnaderna mellan olika granskningstyper: informell granskning, genomgång, teknisk granskning och inspektion

FL-3.2.4 (K3) Tillämpa en granskningsteknik på en arbetsprodukt för att hitta defekter

FL-3.2.5 (K2) Förklara de faktorer som bidrar till en framgångsrik granskning

3.1 Grunder för statisk testning

Som kontrast till dynamisk testning, som kräver exekvering av programvaran som testas, förlitar sig statisk testning på manuell kontroll av arbetsprodukterna (det vill säga granskningar) eller verktygsdriven utvärdering av koden eller andra arbetsprodukter (det vill säga statisk analys). Båda typerna av statisk testning utvärderar koden eller andra arbetsprodukter som testas utan att exekvera koden eller arbetsprodukten som testas.

Statisk analys är viktigt för säkerhetskritiska datorsystem (till exempel programvara för flygplan, medicinsk utrustning eller kärnkraftverk), men statisk analys har också blivit allt viktigare och vanligare i andra miljöer. Statisk analys är till exempel en viktig del av informationssäkerhetstestning. Statisk analys ingår också ofta i automatiserade bygg- och leveranssystem, till exempel vid agil utveckling, kontinuerlig leverans (continuous delivery) och kontinuerlig distribution (continuous deployment).

3.1.1 Arbetsprodukter som kan granskas med statisk testning

Nästan alla arbetsprodukter går att granska med statisk testning (granskningar och/eller statisk analys), till exempel:

- Specifikationer, inklusive verksamhetskrav, funktionella krav och informationssäkerhetskrav
- Epics, användarberättelser och acceptanskriterier
- Arkitektur- och designspecifikationer
- Kod
- Testvara, inklusive testplaner, testfall, testprocedurer och automatiserade testskript
- Användarguider
- Webbsidor
- Kontrakt, projektplaner, tidsplaner och budgetar
- Modeller, till exempel aktivitetsdiagram, som kan användas för modellbaserad testning (se ISTQB-MBT Foundation Level Model-Based Tester Extension Syllabus och Kramer 2016)

Granskningar kan tillämpas på alla arbetsprodukter som deltagarna kan läsa och förstå. Statisk analys kan tillämpas effektivt på alla arbetsprodukter med en formell struktur (normalt sett kod eller modeller) där det finns ett lämpligt verktyg för statisk analys. Statisk analys kan till och med använda verktyg som utvärderar arbetsprodukter skrivna med naturligt språk, som exempelvis krav (till exempel kontroll av stavning, grammatik och läsbarhet).

3.1.2 Fördelar med statisk testning

Tekniker för statisk testning ger en rad fördelar. När statisk testning tillämpas tidigt under programvarans utvecklingslivscykel, möjliggör det tidig upptäckt av defekter innan den dynamiska testningen genomförs (till exempel vid granskning av krav eller designspecifikationer samt förfining av produktbackloggar). Defekter som upptäckts tidigt är ofta mycket billigare att åtgärda än defekter som upptäcks senare under livscykeln, särskilt jämfört med defekter som upptäcks efter att programvaran implementeras och används aktivt. Att använda tekniker för statisk testning för att upptäcka defekter och sedan åtgärda dessa snabbt blir nästan alltid mycket billigare för organisationen än att använda dynamisk testning för att upptäcka dem i testobjektet och sedan åtgärda dem, särskilt med tanke på de ytterligare kostnader som förknippas med att uppdatera andra arbetsprodukter och utföra omtestning och regressionstestning.

Ytterligare fördelar med statisk testning kan omfatta:

- Upptäcka och korrigera defekter mer effektivt, och innan den dynamiska testexekveringen
- Upptäcka defekter som inte är enkla att hitta med dynamisk testning
- Förhindra defekter i design eller kodning genom att upptäcka inkonsekvenser, tvetydigheter, motsägelser, utelämnanden, felaktigheter och redundans i kraven
- Ökad produktivitet i utvecklingen (till exempel tack vare förbättrad design och mer underhållbar kod)
- Minskad utvecklingskostnad och tid
- Minskad tid och kostnad för test
- Minskad total kvalitetskostnad över programvarans livslängd tack vare färre fel senare i livscykeln eller i drift
- Förbättrad kommunikation mellan teammedlemmar som deltar i granskningar

3.1.3 Skillnader mellan statisk och dynamisk testning

Statisk testning och dynamisk testning kan ha samma mål (se avsnitt 1.1.1), till exempel att visa en utvärdering av kvaliteten på arbetsprodukterna och upptäcka defekter så tidigt som möjligt. Statisk och dynamisk testning kompletterar varandra genom att upptäcka olika typer av defekter.

En huvudskillnad är att statisk test upptäcker defekter direkt i arbetsprodukterna istället för att upptäcka felsymptom som orsakas av defekter när programvaran exekveras. En defekt kan existera i en arbetsprodukt under mycket lång tid utan att orsaka ett felsymptom. Den kodgren där defekten finns kanske sällan exekveras eller kan vara svår att nå, vilket gör att det inte är enkelt att utveckla och exekvera ett dynamiskt test som stöter på den. Statisk testning kan hitta defekten med mycket mindre ansträngning.

En annan skillnad är att statisk testning kan användas för att förbättra enhetligheten och den interna kvaliteten hos arbetsprodukterna, medan dynamisk testning normalt sett fokuserar på externt synliga beteenden.

De typiska defekter som går enklare och billigare att hitta och åtgärda genom statisk testning jämfört med dynamisk testning omfattar:

- Kravdefekter (till exempel inkonsekvenser, tvetydigheter, motsägelser, utelämnanden, felaktigheter och redundanser)
- Designdefekter (till exempel ineffektiva algoritmer eller databasstrukturer, hög koppling, liten sammanhållning (high coupling, low cohesion))
- Koddefekter (till exempel variabler med odefinierade värden, variabler som deklareras men aldrig används, onåbar kod och dubblettkod)
- Avvikelser från standarder (till exempel bristande efterlevnad av kodningsstandarder)
- Felaktiga gränssnittsspecifikationer (till exempel användning av olika mätenheter i det anropande systemet och det anropade systemet)
- Säkerhetsproblem (till exempel mottaglighet för buffertöverskridning)
- Luckor eller felaktigheter i testbasens spårbarhet eller täckning (till exempel tester som saknas för acceptanskriterier)

Dessutom kan de flesta typer av defekter som rör underhållbarheten endast hittas genom statisk testning (till exempel felaktig modulering, dåliga möjligheter att återanvända komponenter samt kod som är svår att analysera och ändra utan att införa nya defekter).

3.2 Granskningsprocess

Granskningar kan variera från informella till formella. Informella granskningar kännetecknas av att de inte följer någon definierad process och att de inte producerar någon formell dokumentation. Formella granskningar kännetecknas av delaktighet för teamet, dokumenterade resultat från granskningen samt dokumenterade procedurer för att genomföra granskningen. Formaliteten i en granskningsprocess hänger samman med faktorer som modellen för programvarans utvecklingslivscykel, utvecklingsprocessens mognad, komplexiteten hos arbetsprodukten som ska granskas, eventuella legala eller reglerande krav och/eller behovet av en verifieringskedja.

Granskningens fokus beror på de överenskomna målen (till exempel att hitta defekter, skapa förståelse, utbilda deltagare som testare eller nya teammedlemmar eller att diskutera och fatta beslut genom konsensus).

ISO-standarden (ISO/IEC 20246) innehåller mer djupgående beskrivningar av granskningsprocessen för arbetsprodukter, inklusive roller och granskningstekniker.

3.2.1 Process för granskning av arbetsprodukter

En granskningsprocess består av följande huvudaktiviteter:

Planering

- Definiera omfattningen, vilket omfattar syftet med granskningen, vilka dokument eller delar av dokument som ska granskas och vilka kvalitetsegenskaper som ska utvärderas
- Uppskatta arbetsinsats och tidsram
- Identifiera granskningsegenskaper som granskningstyp med roller, aktiviteter och checklistor
- Välja de personer som ska delta i granskningen och fördela roller
- Definiera startkriterier och avslutskriterier för mer formella granskningstyper (till exempel inspektioner)
- Kontrollera att startkriterierna är uppfyllda (för mer formella granskningstyper)

Initiera granskningen

- Distribuera arbetsprodukten (fysiskt eller elektroniskt) och annat material, till exempel loggformulär, checklistor och relaterade arbetsprodukter
- Förklara omfattning, mål, process, roller och arbetsprodukter för deltagarna
- Besvara eventuella frågor som deltagarna kan ha om granskningen

Individuell granskning (det vill säga individuella förberedelser)

- Granska hela eller delar av arbetsprodukten
- Notera möjliga defekter, rekommendationer och frågor

Kommunicera resultat och analys

- Notera identifierade möjliga defekter (till exempel vid ett granskningsmöte)

- Analysera möjliga defekter och tilldela dem ägarskap och status
- Utvärdera och dokumentera kvalitetsegenskaper
- Utvärdera resultatet av granskningen mot avslutskriterierna för att fatta ett granskningsbeslut (avvisa, större ändringar krävs, godkännande, eventuellt med mindre ändringar)

Åtgärda och rapportera

- Skriva felrapporter för de resultat som kräver ändringar
- Åtgärda de funna defekterna i arbetsprodukten som granskas (görs vanligtvis av författaren)
- Vidarebefordra information om defekter till lämplig person eller lämpligt team (när de upptäcks i en arbetsprodukt som är relaterad till den arbetsprodukt som granskas)
- Uppdatera status för defekter (vid formella granskningar), som eventuellt inkluderar överenskommelse från kommentarens författare
- Samla mätetal (för mer formella granskningstyper)
- Kontrollera att avslutskriterierna är uppfyllda (för mer formella granskningstyper)
- Acceptera arbetsprodukten när avslutskriterierna har uppnåtts

Resultatet av granskningen av en arbetsprodukt varierar beroende på granskningstyp och formalitet enligt beskrivningen i avsnitt 3.2.3.

3.2.2 Roller och ansvarsområden vid en formell granskning

En typisk formell granskning inkluderar rollerna nedan:

Författare

- Skapar de arbetsprodukter som granskas
- Åtgärdar defekter i arbetsprodukten som granskas (vid behov)

Ledning

- Ansvarar för att planera granskningen
- Beslutar om genomförande av granskningar
- Tilldelar personal, budget och tid
- Övervakar kostnadseffektiviteten löpande
- Fattar beslut vid otillräckliga resultat

Facilitator (kallas ofta moderator)

- Säkerställer effektivt genomförande av granskningsmöten
- Medlar vid behov mellan olika synvinklar
- Är oftast den person som ligger bakom en framgångsrik granskning

Granskningsansvarig

- Tar det övergripande ansvaret för granskningen
- Beslutar vem som ska delta och organiserar när och var granskningen ska genomföras

Granskare

- Kan vara ämnesexperter, personer som arbetar i projektet, intressenter som har ett intresse för arbetsprodukten och/eller individer med specifik teknisk bakgrund eller verksamhetsbakgrund
- Identifierar eventuella defekter i arbetsprodukten som granskas
- Kan representera olika områden (till exempel testare, programmerare, användare, operatörer, affärsanalytiker eller användbarhetsexperter)

Sekreterare

- Samlar möjliga defekter som upptäckts under individuell granskning
- Dokumenterar möjliga defekter, öppna frågor och beslut som kommit fram under mötet

I vissa granskningstyper kan en person ha mer än en roll, och de aktiviteter som förknippas med varje roll kan också variera beroende på granskningstyp. Dessutom kan införandet av verktyg som stöd för granskningsprocessen göra att det oftast inte behövs någon sekreterare, framför allt tack vare loggning av defekter, öppna punkter och beslut.

Dessutom kan mer detaljerade roller användas enligt beskrivningen i ISO-standarden (ISO/IEC 20246).

3.2.3 Granskningstyper

Även om granskningar kan användas för olika syften, är ett av huvudmålen att upptäcka defekter. Alla granskningstyper kan bidra till att upptäcka defekter, och den valda granskningstypen bör bland annat vara baserad på projektets behov, tillgängliga resurser, produkttyper och risker, verksamhetsdomän och företagskultur.

Granskningar kan klassificeras enligt olika attribut. Här visas de fyra vanligaste typerna av granskning och deras tillhörande attribut.

Informell granskning (till exempel ”buddy check”, arbete i par, pargranskning)

- Huvudsyfte: upptäcka potentiella defekter
- Eventuella ytterligare syften: generera nya idéer eller lösningar, snabbt lösa mindre problem
- Bygger inte på en formell (dokumenterad) process
- Behöver inte vara ett granskningsmöte
- Kan genomföras av en kollega till författaren (”buddy check”) eller av flera personer
- Resultaten kan dokumenteras
- Användbarheten varierar beroende på granskarna
- Användning av checklistor är valfritt
- Mycket vanlig vid agil utveckling

Genomgång

- Huvudsyfte: hitta defekter, förbättra programvaruprodukten, överväga alternativa implementeringar, utvärdera efterlevnad av standarder och specifikationer
- Möjliga ytterligare syften: utbyta idéer om tekniker och stilarter, utbildning av deltagare, uppnå konsensus
- Individuella förberedelser inför granskningsmötet är valfria

- Granskningsmötet leds normalt sett av arbetsproduktens författare
- Sekreterare är obligatoriskt
- Användning av checklistor är valfritt
- Kan även genomföras i form av scenarion, torrkörningar eller simuleringar
- Loggar och granskningsrapporter för potentiella fel kan produceras
- Genomförandet kan variera från informellt till mycket formellt

Teknisk granskning

- Huvudsyfte: skapa konsensus, upptäcka potentiella defekter
- Möjliga ytterligare syften: utvärdera kvalitet och skapa förtroende för arbetsprodukten, skapa nya idéer, motivera och hjälpa författare att förbättra framtida arbetsprodukter, överväga alternativa implementeringar
- Granskare bör vara tekniska kollegor till författaren och tekniska experter inom samma eller andra områden
- Individuella förberedelser inför granskningsmötet krävs
- Granskningsmötet leds med fördel av en utbildad facilitator (normalt sett inte författaren)
- Sekreterare är obligatoriskt, helst inte författaren
- Användning av checklistor är valfritt
- Vanligtvis produceras loggar och granskningsrapporter för potentiella defekter

Inspektion

- Huvudsyften: upptäcka eventuella defekter, utvärdera kvalitet och skapa förtroende för arbetsprodukten samt förhindra liknande defekter i framtiden genom återkoppling till författaren och av grundorsaksanalys
- Möjliga ytterligare syften: motivera och hjälpa författare att förbättra framtida arbetsprodukter och processen för utveckling av programvara, skapa konsensus
- Följer en definierad process med formella dokumenterade utdata som bygger på regler och checklistor
- Använder tydligt definierade roller, till exempel de roller som anges i avsnitt 3.2.2 och som är obligatoriska, och kan inkludera en särskild läsare (som läser arbetsprodukten högt under granskningsmötet)
- Individuella förberedelser inför granskningsmötet krävs
- Granskare är antingen kollegor till författaren eller experter på andra områden som är relevanta för arbetsprodukten
- Angivna start- och avslutskriterier används
- Sekreterare är obligatoriskt
- Granskningsmötet leds av en utbildad facilitator (inte författaren)
- Författaren får inte fungera som granskningsledare, läsare eller sekreterare

- Loggar och granskningsrapporter för potentiella fel produceras
- Mätetal samlas in och används för att förbättra hela utvecklingsprocessen för programvaran, inklusive granskningsprocessen

En enda arbetsprodukt kan vara föremål för mer än en typ av granskning. Om mer än en typ av granskning används kan ordningen variera. En informell granskning kan till exempel genomföras innan en teknisk granskning för att säkerställa att arbetsprodukten är redo för en teknisk granskning.

De typer av granskningar som beskrivs ovan kan genomföras som kollegial granskning, det vill säga av kollegor på ungefär samma organisationsnivå.

Vilka typer av defekter som upptäcks i en granskning varierar, framför allt beroende på vilken arbetsprodukt som granskas. Se avsnitt 3.1.3 för exempel på defekter som går att upptäcka i olika arbetsprodukter, och se Gilb 1993 för information om formella inspektioner.

3.2.4 Tillämpning av granskningstekniker

Det finns en rad olika granskningstekniker som kan tillämpas under den individuella granskningen (det vill säga individuella förberedelser) för att upptäcka defekter. Dessa tekniker kan användas med alla granskningstyper som beskrivs ovan. Teknikernas effektivitet kan variera beroende på vilken typ av granskning som används. Exempel på olika individuella granskningstekniker för olika granskningstyper visas nedan.

Ad hoc

I en ad hoc-granskning får granskarna få eller inga instruktioner om hur uppgiften ska utföras. Granskarna läser ofta arbetsprodukten sekventiellt och identifierar och dokumenterar problemen när de uppstår. Ad hoc-granskning är en ofta använd teknik som inte behöver mycket förberedelser. Den här tekniken är i hög grad beroende av granskarnas färdigheter och kan leda till att samma problem rapporteras av olika granskare.

Checklistebaserad

En checklistebaserad granskning är en systematisk teknik där granskarna identifierar problem baserat på checklistor som delas ut när granskningen inleds (till exempel av facilitatorn). En granskningschecklista består av en uppsättning frågor som bygger på potentiella defekter, och som kan härledas från tidigare erfarenheter. Checklistor bör vara specifika för den typ av arbetsprodukt som granskas och bör underhållas regelbundet för att täcka olika typer av problem som har missats under föregående granskningar. Huvudfördelen med en checklistebaserad teknik är den systematiska täckningsgraden för vanliga typer av defekter. Försiktighet bör vidtas och inte bara följa checklistan vid individuell granskning, utan att också titta efter defekter utanför checklistan.

Scenarion och torrkörningar

I en scenariobaserad granskning förses granskarna med strukturerade riktlinjer för hur arbetsprodukten ska läsas igenom. En scenariobaserad metod hjälper granskarna att genomföra ”torrkörningar” på arbetsprodukterna baserat på den förväntade användningen av arbetsprodukten (om arbetsprodukten är dokumenterad i lämpligt format, till exempel användningsfall). Dessa scenarion ger granskaren bättre riktlinjer för hur specifika typer av defekter ska identifieras i stället för enkla poster i en checklista. Precis som vid checklistebaserad granskning bör granskarna inte vara begränsade till de dokumenterade scenariona så att de inte missar andra typer av defekter (till exempel saknade funktioner).

Rollbaserad

En rollbaserad granskning är en teknik där granskarna utvärderar arbetsprodukten ur de enskilda intressenternas perspektiv. Typiska roller omfattar specifika typer av slutanvändare (erfarna, oerfarna,

seniorer, barn o.s.v.), och specifika roller i organisationen (användaradministratör, systemadministratör, prestandatestare o.s.v).

Perspektivbaserad

Vid perspektivbaserad läsning använder granskarna precis som vid rollbaserad granskning de olika intressenternas synvinkel vid individuell granskning. Typiska synvinklar för intressenterna är slutanvändare, marknadsförare, designer, testare eller driftspersonal. Att använda olika intressenters synvinklar leder till större djup vid individuell granskning och mindre upprepning av samma problem från granskarna.

Dessutom kräver perspektivbaserad läsning också att granskarna försöker använda arbetsprodukten som granskas för att generera den produkt de ska härleda från den. En testare kan till exempel försöka generera utkast till acceptanstester under perspektivbaserad läsning av en kravspecifikation för att se om all nödvändig information inkluderats. Dessutom förväntas det att checklistor används vid perspektivbaserad läsning.

Empiriska studier har visat att perspektivbaserad läsning är den mest effektiva allmänna tekniken för att granska krav och tekniska arbetsprodukter. En viktig framgångsfaktor är att inkludera och väga olika intressenters synvinklar på lämpligt sätt, baserat på risker. I Shul 2000 finns mer information om perspektivbaserad läsning, och i Sauer 2000 finns information om effektiviteten hos olika granskningstyper.

3.2.5 Framgångsfaktorer för granskningar

För att genomföra en lyckad granskning måste rätt typ av granskning och tekniker användas. Dessutom finns det ett antal andra faktorer som påverkar utgången av granskningen.

De organisatoriska framgångsfaktorerna för granskningar omfattar:

- Varje granskning har tydliga mål som definieras under granskningsplaneringen och används som mätbara avslutskriterier
- De granskningstyper som används ska vara lämpliga för att uppnå målen och passa för typen av arbetsprodukter och deltagarnas kunskapsnivå
- Alla granskningstekniker som används, till exempel checklistebaserad eller rollbaserad granskning, är lämpliga för att identifiera defekter i arbetsprodukten som granskas
- Alla checklistor som används tar upp huvudriskerna och är uppdaterade
- Stora dokument skrivs och granskas i små delar, så att kvalitetskontrollen omfattar att tidigt och ofta ge feedback till författarna om defekter
- Deltagarna har tillräckligt med tid på sig till förberedelser
- Granskningar planeras i god tid
- Ledningen stöttar granskningsprocessen (till exempel genom att skapa tillräckligt med tid för granskningsaktiviteter i projekttidsplanerna)

Personrelaterade framgångsfaktorer för granskningar omfattar:

- Rätt personer deltar för att uppfylla granskningsmålen, till exempel personer med olika färdigheter eller perspektiv som kan använda dokumentet som indata för arbetet
- Testarna ses som värdefulla granskare som bidrar till granskningen och lär sig mer om arbetsprodukten, så att de kan förbereda mer effektiva tester i ett tidigare skede

- Deltagarna lägger tillräckligt med tid och uppmärksamhet på detaljerna
- Granskningar genomförs på små delar så att granskarna inte tappar koncentrationen under den enskilda granskningen och/eller granskningsmötet
- Defekter som upptäcks bekräftas, bedöms och hanteras objektivt
- Mötet leds på ett bra sätt så deltagarna inte upplever att det är slöseri med tid
- Granskningen genomförs i en anda av förtroende: resultatet kommer inte att användas för att utvärdera deltagarna
- Deltagarna undviker kroppsspråk och beteende som kan tyda på uttråkning, irritation eller fientlighet mot de andra deltagarna
- Tillräcklig utbildning tillhandahålls, särskilt vid mer formella granskningstyper som inspektioner
- En kultur av lärande och processförbättringar främjas

Se Gilb 1993, Wiegers 2002 och van Veenendaal 2004 för mer information om framgångsrika granskningar.

4 Testtekniker — 330 minuter

Nyckelord

användningsfallsbaserad testning, beslutstäckning, black-box-testteknik, checklistebaserad testning, ekvivalensklassindelning, erfarenhetsbaserad testteknik, felgissning, gränsvärdesanalys, kodsatstäckning, testning med hjälp av beslutstabeller, testteknik, tillståndsbaserad testning, täckningsgrad, utforskande täckning, white-box-testteknik

Utbildningsmål för testtekniker

4.1 Kategorier för testtekniker

FL-4.1.1 (K2) Förklara egenskaper, likheter och skillnader mellan black-box-testtekniker, white-box-testtekniker och erfarenhetsbaserade testtekniker

4.2 Black-box-testtekniker

FL-4.2.1 (K3) Tillämpa ekvivalensklassindelning för att härleda testfall från givna krav

FL-4.2.2 (K3) Tillämpa gränsvärdesanalys för att härleda testfall från givna krav

FL-4.2.3 (K3) Tillämpa testning med hjälp av beslutstabeller för att härleda testfall från givna krav

FL-4.2.4 (K3) Tillämpa tillståndsbaserad testning för att härleda testfall från givna krav

FL-4.2.5 (K2) Förklara hur testfall kan härledas från ett användningsfall

4.3 White-box-testtekniker

FL-4.3.1 (K2) Förklara kodsatstäckning

FL-4.3.2 (K2) Förklara beslutstäckning

FL-4.3.3 (K2) Förklara värdet av beslutstäckning och kodsatstäckning

4.4 Erfarenhetsbaserade testtekniker

FL-4.4.1 (K2) Förklara felgissning

FL-4.4.2 (K2) Förklara utforskande testning

FL-4.4.3 (K2) Förklara checklistebaserad testning

4.1 Kategorier för testtekniker

Syftet med en testteknik, inklusive de som diskuteras i det här avsnittet, är att ge stöd vid identifierandet av testvillkor, testfall och testdata.

4.1.1 Välja testtekniker

Vilka testtekniker som ska användas beror på en rad faktorer, som till exempel:

- Typ av komponent eller system
- Komponentens eller systemets komplexitet
- Reglerande standarder
- Kundkrav eller kontraktsmässiga krav
- Risknivåer
- Risktyper
- Testmål
- Tillgänglig dokumentation
- Testarnas kunskaper och färdigheter
- Tillgängliga verktyg
- Tid och budget
- Programvarans utvecklingslivscykelmodel
- Förväntad användning av programvaran
- Tidigare erfarenhet av att använda testteknikerna för komponenten eller systemet som ska testas
- Vilka typer av defekter som förväntas i komponenten eller systemet

Vissa testtekniker är mer användbara för vissa situationer och testnivåer. Andra testtekniker kan tillämpas på alla testnivåer. När testfallen skapas använder testarna i allmänhet en kombination av testtekniker för att uppnå bästa resultat av testarbetet.

Användning av testtekniker i aktiviteter för testanalys, testdesign och testimplementering kan variera från mycket informell (lite eller ingen dokumentation) till mycket formell. Den lämpliga formalitetsnivån beror på testningens sammanhang, inklusive mognaden för test- och utvecklingsprocesserna, tidsbegränsningar, säkerhetskrav eller reglerande krav, kunskaper och färdigheter hos inblandade personer och vilken modell för programvarans utvecklingslivscykel som används.

4.1.2 Kategorier av testtekniker och deras egenskaper

I den här kursplanen klassificeras testteknikerna som black-box, white-box eller erfarenhetsbaserade.

Black-box-testtekniker (kallas även beteendebaserade tekniker) bygger på en analys av en lämplig testbas (till exempel dokument med formella krav, specifikationer, användningsfall, användarberättelser eller affärsprocesser). Dessa tekniker kan tillämpas för både funktionell och icke-funktionell testning. Black-box-testtekniker koncentrerar sig på indata och utdata för testobjektet utan att ta hänsyn till den interna strukturen.

White-box-testtekniker (kallas även strukturella eller strukturbaserade tekniker) bygger på en analys av arkitekturen, detaljerad design, intern struktur eller testobjektets kod. Till skillnad från black-box-testtekniker koncentreras white-box-testtekniker på strukturen och behandlingen inom testobjektet.

Erfarenhetsbaserade testtekniker utnyttjar erfarenheten hos utvecklare, testare och användare för att designa, implementera och exekvera tester. Dessa tekniker kombineras ofta med black-box- och white-box-testtekniker.

Vanliga egenskaper för black-box-testtekniker omfattar:

- Testvillkor, testfall och testdata härleds från en testbas som kan vara programvarukrav, specifikationer, användningsfall och användarberättelser
- Testfall kan användas för att upptäcka skillnader mellan kraven och implementeringen av kraven, men även avvikelser från kraven
- Täckningsgraden mäts baserat på de objekt som testas i testbasen och den teknik som tillämpas för testbasen

Vanliga egenskaper för white-box-testtekniker omfattar:

- Testvillkor, testfall och testdata härleds från en testbas som kan inkludera kod, programvarans arkitektur, detaljerad design eller andra informationskällor som rör programvarans struktur
- Täckningsgraden mäts baserat på de objekt som testas inom en vald struktur (till exempel koden eller gränssnitten)
- Specifikationer används ofta som en extra informationskälla för att avgöra det förväntade resultatet av testfallen

Vanliga egenskaper för erfarenhetsbaserade testtekniker omfattar:

- Testvillkor, testfall och testdata härleds från en testbas som kan omfatta kunskaper och erfarenheter hos testare, utvecklare, användare och andra intressenter

Dessa kunskaper och erfarenheter omfattar förväntad användning av programvaran, dess miljö, potentiella defekter och fördelningen av dessa defekter.

Den internationella standarden (ISO/IEC/IEEE 29119-4) innehåller beskrivningar av testtekniker och deras motsvarande åtgärder för täckningsgrad (se Craig 2002 och Copeland 2004 för mer information om teknikerna).

4.2 Black-box-testtekniker

4.2.1 Ekvivalensklassindelning

Ekvivalensklassindelning delar upp data i partitioner (kallas även ekvivalensklasser) på ett sätt som gör att alla medlemmar i en viss klass förväntas behandlas på samma sätt (se Kaner 2013 och Jorgensen 2014). Det finns ekvivalensklasser för både giltiga och ogiltiga värden.

- Giltiga värden är värden som bör godkännas av komponenten eller systemet. En ekvivalensklass som innehåller giltiga värden kallas för en ”giltig ekvivalensklass”.
- Ogiltiga värden är värden som bör avvisas av komponenten eller systemet. En ekvivalensklass som innehåller ogiltiga värden kallas för en ”ogiltig ekvivalensklass”.
- Klasserna kan identifieras för alla dataelement som rör testobjektet, inklusive indata, utdata, interna värden, tidsrelaterade värden (till exempel före eller efter en händelse) och för

gränssnittsparametrar (till exempel integrerade komponenter som testas under integrationstestning).

- Alla klasser kan vid behov delas in i underklasser.
- Varje värde får endast tillhöra en enda ekvivalensklass.
- När ogiltiga ekvivalensklasser används i testfall bör de testas individuellt, det vill säga inte kombineras med andra ekvivalensklasser, för att säkerställa att felsymptomen inte maskeras. Felsymptomen kan vara maskerade när flera felsymptom inträffar samtidigt, men bara ett är synligt och gör att de andra inte upptäcks.

För att uppnå en täckningsgrad på 100 % med den här tekniken måste testfallen täcka alla identifierade klasser (inklusive ogiltiga klasser) genom att använda minst ett värde från varje klass. Täckningsgraden mäts som antalet ekvivalensklasser som testas med minst ett värde, delat med det totala antalet identifierade ekvivalensklasser, normalt sett uttryckt i procent. Ekvivalensklassindelning kan tillämpas på alla testnivåer.

4.2.2 Gränsvärdesanalys

Gränsvärdesanalys är en utvidgning av ekvivalensklassindelning, men går endast att använda när klassen är i ordningsföljd och består av numeriska eller sekventiella data. De lägsta och högsta värdena (eller första och sista värdena) i en klass är dess gränsvärden (Beizer 1990).

Antag till exempel att ett indatafält tar emot ett enda heltalsvärde som indata, med hjälp av en knappsats som begränsar inmatningarna så att inga andra indata än heltal går att ange. Det giltiga området är från 1 till 5. Det finns alltså tre ekvivalensklasser: ogiltig (för låg), giltig och ogiltig (för hög). För den giltiga ekvivalensklassen är gränsvärdena 1 och 5. För den ogiltiga ekvivalensklassen (för hög) är gränsvärdena 6 och 9. För den ogiltiga ekvivalensklassen (för låg) finns det endast ett gränsvärde, 0, eftersom den här klassen endast har en medlem.

I exemplet ovan identifierar vi två gränsvärden per gräns. Gränsen mellan ogiltiga (för låga) och giltiga värden ger testvärdena 0 och 1. Gränsen mellan giltiga och ogiltiga (för höga) värden ger testvärdena 5 och 6. En del variationer av den här tekniken identifierar tre gränsvärden per gräns: värdena före, på och precis över gränsen. Om vi använder gränsvärden med tre värden i föregående exempel är testvärdena för det lägre gränsvärdet 0, 1 och 2 och testvärdena för de övre gränsvärdena 4, 5 och 6 (Jorgensen 2014).

Programvarans beteende vid ekvivalensklassernas gränser löper större risk att vara felaktiga än för värden inom klasserna. Det är viktigt att komma ihåg att både specificerade och implementerade gränser kan förskjutas till positioner över eller under deras avsedda positioner, utelämnas helt och hållet eller kompletteras med ytterligare oönskade gränser. Gränsvärdesanalys och testning avslöjar nästan alla sådana defekter genom att tvinga programvaran att visa beteenden från en annan klass än den som gränsvärdet borde tillhöra.

Gränsvärdesanalys kan tillämpas på alla testnivåer. Den tekniken används i allmänhet för att testa krav som kräver ett intervall med tal (inklusive datum och tider). Gränsvärdestäckningen för en klass mäts som antalet gränsvärden som testats delat med det totala antalet specificerade testvärden, normalt uttryckt som ett procenttal.

4.2.3 Testning med hjälp av beslutstabeller

Kombinatoriska testtekniker är användbara för att testa implementering av systemkrav som anger hur olika kombinationer av villkor resulterar i olika resultat. En metod för sådan testning är testning med hjälp av beslutstabeller.

Beslutstabeller är ett bra sätt att registrera komplexa verksamhetsregler som ett system måste implementera. När beslutstabeller skapas identifierar testaren villkor (ofta indata) och de resulterande åtgärderna (ofta utdata) för systemet. Dessa utgör rader i tabellen, vanligtvis med villkoren längst upp och åtgärderna längst ned. Varje kolumn motsvarar en beslutsregel som definierar en unik kombination av villkor som resulterar i exekveringen av de åtgärder som är förknippade med regeln. Värdena på villkoren och åtgärderna visas vanligtvis som booleska värden (sant eller falskt) eller diskreta värden (till exempel rött, grönt eller blått), men kan också vara tal eller intervall. Dessa olika typer av villkor och åtgärder kan finnas tillsammans i samma tabell.

Den gemensamma notationen i beslutstabellerna är enligt följande:

För villkor:

- Y innebär att villkoret är sant (kan också visas som T eller 1)
- N innebär att villkoret är falskt (kan också visas som F eller 0)
- — innebär att värdet på villkoret inte har någon betydelse (kan också visas som N/A)

För åtgärder:

- X innebär att åtgärden ska inträffa (kan också visas som Y, T eller 1)
- Tom innebär att åtgärden inte ska inträffa (kan också visas som –, N, F eller 0)

Ett förtydligande: Y står för Yes (Ja), T för True (Sant) och F för False (Falskt).

En komplett beslutstabell har tillräckligt med kolumner för att täcka alla kombinationer av villkor. Tabellen kan minskas genom att ta bort kolumner som innehåller omöjliga kombinationer av villkor, kolumner som innehåller möjliga men onåbara kombinationer av villkor samt kolumner som testar kombinationer av villkor som inte påverkar resultatet. Mer information om hur beslutstabeller minskas finns i ISTQB-ATA Advanced Level Test Analyst Syllabus.

Den minsta gemensamma standarden för täckningsgraden vid testning med hjälp av beslutstabeller är att ha minst ett testfall per beslutsregel i tabellen. Detta innebär normalt att alla kombinationer av villkor är täckta. Täckningsgraden mäts som antalet beslutsregler som testas med minst ett testfall, delat med det totala antalet beslutsregler, normalt sett uttryckt i procent.

Styrkan i testning med hjälp av beslutstabeller är att den bidrar till att identifiera alla viktiga kombinationer av villkor, där en del annars kan bli förbisedda. Dessutom bidrar den till att hitta eventuella luckor i kraven. Den kan tillämpas i alla situationer där programvarans beteende beror på en kombination av villkor, och på alla testnivåer.

4.2.4 Tillståndsbaserad testning

Komponenter eller system kan reagera olika på en händelse beroende på aktuella villkor eller tidigare historik (till exempel de händelser som har inträffat sedan systemet initialiserades). Den tidigare historiken kan sammanfattas med hjälp av tillståndskonceptet. Ett tillståndsövergångsdiagram visar de möjliga programvarutillstånden samt hur programvaran startar, avslutar och övergår mellan tillstånden. Ett tillstånd initieras av en händelse (till exempel att användaren anger ett värde i ett fält). Händelsen resulterar i en övergång. Om samma händelse kan resultera i två eller flera övergångar från samma

tillstånd, kan händelsen kvalificeras genom ett begränsningsvillkor. Tillståndsändringen kan resultera i att programvaran vidtar en åtgärd (till exempel visar en beräkning eller ett felmeddelande).

En tillståndsövergångstabell visar alla giltiga övergångar mellan tillstånden, samt händelser, begränsningsvillkor och resulterande åtgärder för giltiga övergångar. Tillståndsövergångsdiagram visar normalt endast de giltiga övergångarna och utelämnar de ogiltiga övergångarna.

Tester kan designas för att täcka en typisk ordningsföljd av tillstånd, att använda alla tillstånd, att utöva varje övergång, att använda specifika sekvenser av övergångar eller att testa ogiltiga övergångar.

Tillståndsbaserad testning används för menybaserade applikationer och används i stor utsträckning inom verksamheter för inbyggd programvara. Tekniken är också lämplig för att modellera ett verksamhetsscenario som har specifika tillstånd eller för att testa skärmnavigering. Konceptet med ett tillstånd är abstrakt – det kan representera ett par rader med kod eller en hel affärsprocess.

Täckningsgraden mäts normalt sett som antalet identifierade tillstånd eller övergångar som testas, delat med antalet identifierade tillstånd eller övergångar i testobjektet, normalt sett uttryckt i procent. Mer information om kriterier för täckningsgrad för tillståndsbaserad testning finns i ISTQB-ATA Advanced Level Test Analyst Syllabus.

4.2.5 Användningsfallsbaserad testning

Tester kan härledas från användningsfall, som är ett specifikt sätt att designa samarbete med programvaruobjekt och ta med kraven för de programvarufunktioner som representeras av användningsfallen. Användningsfall är förknippade med aktörer (mänskliga användare, extern maskinvara eller andra komponenter eller system) och subjekt (den komponent eller det system som användningsfallet är avsett för).

Varje användningsfall anger ett beteende som ett subjekt kan genomföra i samarbete med en eller flera aktörer (UML 2.5.1 2017). Ett användningsfall kan beskrivas genom växelverkan och aktiviteter, samt med förutsättningar, sluttillstånd och naturliga språk när så är lämpligt. Växelverkan mellan aktörerna och subjektet kan resultera i ändringar av subjektets tillstånd. Växelverkan kan representeras grafiskt med arbetsflöden, aktivitetsdiagram eller affärsprocessmodeller.

Ett användningsfall kan innehålla möjliga variationer i sitt grundbeteende (huvudscenario), inklusive exceptionellt beteende (alternativa scenarion) och felhantering (systemsvar och återhämtning från programmerings-, tillämpnings- och kommunikationsfel som till exempel kan resultera i ett felmeddelande). Tester designas för att använda de definierade beteendena (huvudscenario, alternativa scenarion samt felhantering). Täckningsgraden kan mätas genom antal av användningsfallens beteenden som testats delat med det totala antalet beteenden, normalt sett uttryckt i procent.

Mer information om kriterier för täckningsgrad vid användningsfallsbaserad testning finns i ISTQB-ATA Advanced Level Test Analyst Syllabus.

4.3 White-box-testtekniker

White-box-testning bygger på testobjektets interna struktur. White-box-testtekniker kan användas på alla testnivåer, men de två kodrelaterade tekniker som diskuteras i det här avsnittet används vanligtvis på komponenttestnivå. Det finns mer avancerade tekniker som används i vissa funktionssäkerhetskritiska miljöer, uppdragskritiska miljöer eller miljöer med hög integritet för att uppnå en mer grundlig täckningsgrad, men dessa diskuteras inte här. Mer information om sådana tekniker finns i ISTQB Advanced Level Technical Test Analyst Syllabus.

4.3.1 Kodsatstestning och täckning

Kodsatstestning testar de exekverbara satserna i koden. Täckningsgraden mäts normalt sett som antalet kodsatser som exekveras av testerna delat med antalet exekverbara kodsatser i testobjektet, normalt sett uttryckt i procent.

4.3.2 Beslutstestning och täckning

Vid beslutstestning skapas testfall som exekverar utpekade beslutsutfall i koden och sedan testar att koden som exekveras bygger på beslutsutfallen. För att göra detta följer testfallen de kontrollflöden som uppstår från en beslutspunkt (till exempel för en IF-sats, ett för det sanna utfallet och ett för det falska utfallet, för en CASE-sats skulle testfall krävas för alla möjliga utfall, inklusive standardutfallet).

Täckningsgraden mäts normalt sett som antalet beslutsutfall som exekveras av testerna delat med antalet exekverbara beslutsutfall, normalt sett uttryckt i procent.

4.3.3 Värdet av kodsatstestning och beslutstestning

När 100 % kodsatstäckning har uppnåtts säkerställs att alla exekverbara satser i koden har testats minst en gång, men det säkerställer inte att all beslutslogik har testats. Av de två white-box-tekniker som diskuteras i den här kursplanen kan kodsatstestning ge lägre täckningsgrad än beslutstestning.

När 100 % beslutstäckning uppnås exekveras samtliga beslutsutfall, vilket inkluderar att testa det sanna utfallet och det falska utfallet även när det inte finns något uttryckligt falskt utfall (till exempel för en IF-sats utan ett ”else” i koden). Kodsatstäckning hjälper till att hitta defekter i koden som inte exekverats av andra tester. Beslutstäckning hjälper till att hitta defekter i kod där andra tester inte har tagit med både sanna och falska utfall.

Att uppnå 100 % beslutstäckning garanterar 100 % kodsatstäckning (men inte tvärt om).

4.4 Erfarenhetsbaserade testtekniker

Vid användning av erfarenhetsbaserade testtekniker härleds testfallen från testarnas färdigheter och intuition, samt deras erfarenhet av liknande applikationer och tekniker. Dessa tekniker kan vara nyttiga för att identifiera tester som inte enkelt kan identifieras genom mer systematiska tekniker. Beroende på testarens angreppssätt och erfarenhet kan dessa tekniker ge mycket varierande täckningsgrader och effektivitet. Täckningsgraden kan vara svår att bedöma och går kanske inte att mäta med de här teknikerna.

Vanliga erfarenhetsbaserade tekniker diskuteras i kommande avsnitt.

4.4.1 Felgissning

Felgissning är en teknik som används för att förutse förekomsten av misstag, defekter och felsymptom baserat på testarens kunskaper, inklusive:

- Hur applikationen har fungerat tidigare
- Vilka typer av misstag som utvecklarna brukar göra
- Felsymptom som har uppstått i andra applikationer

En metodisk inställning till felgissningstekniken är att skapa en lista över möjliga misstag, defekter och felsymptom och utforma tester som avslöjar dessa felsymptom och de defekter som orsakar dem. Dessa listor med misstag, defekter och felsymptom kan baseras på erfarenhet, uppgifter om defekter och felsymptom, eller via kunskaper om vanliga orsaker till att programvara inte fungerar.

4.4.2 Utforskande testning

Utforskande testning går ut på att dynamiskt under testexekveringen designa, exekvera, logga och utvärdera informella tester (ej fördefinierade). Testresultaten används för att ta lära sig mer om komponenten eller systemet, och för att skapa tester för de områden som kan behöva mer testning.

Utforskande testning genomförs ibland som sessionsbaserad testning för att strukturera aktiviteterna. Vid sessionsbaserad testning genomförs utforskande testning inom en definierad tidsram, och testaren använder en testcharter som innehåller testmål för att styra testningen. Testaren kan använda testsessionsblad för att dokumentera de steg som följs och de upptäckter som görs.

Utforskande testning är som mest användbar när det finns få eller otillräckliga specifikationer eller extrem tidspress under testningen. Utforskande testning är också användbart för att komplettera andra mer formella testtekniker.

Utforskande testning är starkt förknippad med reaktiva teststrategier (se avsnitt 5.2.2). Utforskande testning kan inkludera användning av andra black-box-tekniker, white-box-tekniker och erfarenhetsbaserade tekniker.

4.4.3 Checklistebaserad testning

Vid checklistebaserad testning designar, implementerar och exekverar testarna tester som täcker de testvillkor som finns i en checklista. Som del av analysen skapar testarna en ny checklista eller utökar en befintlig checklista, men man kan också använda en befintlig checklista utan ändringar. Sådana checklistor kan skapas baserat på erfarenhet, kunskaper om vad som är viktigt för användaren eller förståelse för varför och hur fel uppstår i programvaran.

Checklistor kan skapas som stöd för olika testtyper, inklusive funktionell och icke-funktionell testning. I avsaknad av detaljerade testfall kan checklistebaserad testning ge riktlinjer och en högre grad av konsistens. Eftersom dessa listor är på hög nivå uppstår troligtvis en del variationer i själva testningen, vilket kan ge större täckningsgrad och mindre repeterbarhet.

5 Testledning — 225 minuter

Nyckelord

angreppssätt för test, avslutskriterier, konfigurationshantering, felhantering, startkriterier, produktrisk, projektrisk, risk, risknivå, riskbaserad testning, sammanfattande testrapporter, testare, testledare, testplan, testplanering, teststatusrapporter, teststrategi, teststyrning, testuppskattning, testövervakning

Utbildningsmål för testledning

5.1 Testorganisation

FL-5.1.1 (K2) Förklara fördelarna och nackdelarna med oberoende testning

FL-5.1.2 (K1) Komma ihåg uppgifterna för en testledare och en testare

5.2 Testplanering och testuppskattning

FL-5.2.1 (K2) Sammanfatta syftet och innehållet i en testplan

FL-5.2.2 (K2) Klargöra skillnader mellan olika teststrategier

FL-5.2.3 (K2) Ge exempel på möjliga start- och avslutskriterier

FL-5.2.4 (K3) Tillämpa kunskaper om prioritering samt tekniska och logiska beroenden för att schemalägga testexekvering för en given uppsättning testfall

FL-5.2.5 (K1) Känna igen faktorer som påverkar arbetsinsatsen vid testning

FL-5.2.6 (K2) Förklara skillnaderna mellan två beräkningstekniker: mätetalsbaserad teknik och expertbaserad teknik

5.3 Testövervakning och teststyrning

FL-5.3.1 (K1) Komma ihåg mätetal som används för testning

FL-5.3.2 (K2) Sammanfatta syftet, innehållet och målgruppen för testrapporter

5.4 Konfigurationshantering

FL-5.4.1 (K2) Sammanfatta hur konfigurationshantering stödjer testning

5.5 Risker och testning

FL-5.5.1 (K1) Definiera risknivåer genom att använda sannolikhet och påverkan

FL-5.5.2 (K2) Förklara skillnaden mellan projektrisker och produktrisker

FL-5.5.3 (K2) Beskriv med hjälp av exempel hur produktriskanalyser kan påverka testningens grundlighet och omfattning

5.6 Felhantering

FL-5.6.1 (K3) Skriva en felrapport som tar upp defekter som har upptäckts under testningen

5.1 Testorganisation

5.1.1 Oberoende testning

Testarbete kan utföras av personer som har en specifik testroll, eller av personer i en annan roll (till exempel kunder). En viss grad av oberoende gör ofta att testaren mer effektivt kan hitta defekter på grund av skillnader mellan författarens och testarens förutfattade meningar (se avsnitt 1.5). Oberoende kan dock inte ersätta förtrogenhet, och utvecklare kan effektivt hitta många defekter i sin egen kod.

Grader av oberoende i testningen inkluderar följande (från låg grad av oberoende till hög grad):

- Inga oberoende testare: den enda formen av testning som finns tillgänglig är utvecklare som testar sin egen kod
- Oberoende utvecklare eller testare i utvecklingsteamen eller projektteamet; det kan vara utvecklare som testar sina kollegors produkter
- Oberoende testteam eller testgrupper inom organisationen som rapporterar till projektledningen eller den exekutiva ledningen
- Oberoende testare från verksamhetsorganisationen eller användargrupper, eller med specialistkunskaper inom specifika testtyper, till exempel användbarhet, informationssäkerhet, prestanda, regulatorisk efterlevnad eller portabilitet
- Oberoende inhyrda testare som inte ingår i organisationen och som antingen arbetar på plats (insourcing) eller på annan plats (outsourcing)

För de flesta typer av projekt är det vanligtvis bäst att ha flera testnivåer, där vissa av dessa nivåer hanteras av oberoende testare. Utvecklare bör delta i testningen, framför allt på lägre nivåer, så att de kan påverka kvaliteten på sitt eget arbete.

Hur den oberoende testningen implementeras varierar beroende på modellen för programvarans utvecklingslivscykel. Vid agil utveckling kan testarna till exempel ingå i ett utvecklingsteam. I vissa organisationer som använder agila metoder kan dessa testare även anses vara del av ett större oberoende testteam. I sådana organisationer kan produktägarna dessutom utföra acceptanstestning för att validera användarberättelserna i slutet av varje iteration.

Möjliga fördelar med oberoende testning omfattar:

- Oberoende testare har större sannolikhet att identifiera olika typer av felsymptom jämfört med utvecklarna på grund av olika bakgrunder, tekniska perspektiv och fördomar
- En oberoende testare kan verifiera, utmana eller motbevisa antaganden som görs av intressenter under specifikation och implementering av systemet

Möjliga nackdelar med oberoende testning omfattar:

- Isolering från utvecklingsteamet, vilket leder till en brist på samarbete, fördröjningar i återkopplingen till utvecklingsteamet eller fientlig stämning gentemot utvecklingsteamet
- Utvecklarna kan förlora ansvarskänslan för kvaliteten
- Oberoende testare kan ses som en flaskhals eller få skulden för försenade releaser
- Oberoende testare kan sakna viktig information (till exempel om ett testobjekt)

Många organisationer lyckas uppnå fördelarna med oberoende testning samtidigt som de undviker nackdelarna.

5.1.2 Uppgifter för testledare och testare

Den här kursplanen tar upp två testroller, testledare och testare. Vilka aktiviteter och uppgifter som utförs av dessa två roller beror på projektets och produktens sammanhang, färdigheterna hos personerna i de olika rollerna och organisationen.

Testledaren har övergripande ansvar för testprocessen och att leda testaktiviteterna framgångsrikt. Testledningsrollen kan utföras av en professionell testledare eller av en projektansvarig, en utvecklingschef eller någon med ansvar för kvalitetssäkring. I större projekt eller organisationer kan flera testteam rapportera till en testledare, testcoach eller testsamordnare, och varje team leds av en testledare eller ansvarig testare. En testledare kan i vissa organisationer benämnas testansvarig.

Typiska uppgifter för testledare kan omfatta:

- Utveckla eller granska en testpolicy eller teststrategi för organisationen
- Planera testaktiviteterna med hänsyn till sammanhanget samt förstå testmålen och riskerna. Detta kan vara att välja testangreppssätt, beräkna testtid, arbetsinsats och kostnad, anskaffa resurser, definiera testnivåer och testcykler samt planera felhantering
- Skriva och uppdatera testplanen/-planerna
- Samordna testplanerna med projektledarna, produktägarna och andra
- Dela testperspektiv med andra projektaktiviteter, till exempel integrationsplanering
- Initiera analysen, designen, implementationen och exekveringen av testerna, övervaka teststatusen och resultaten och kontrollera statusen för avslutskriterierna (eller ”definition of done”)
- Förbereda och leverera teststatusrapporter och sammanfattade testrapporter baserade på den insamlade informationen
- Anpassa planeringen baserat på testresultat och teststatus (som ibland dokumenteras i teststatusrapporter och/eller i sammanfattade testrapporter för annan testning som redan slutförts i projektet) och vidta eventuella åtgärder som är nödvändiga för teststyrning
- Stödja införandet av felhanteringssystemet och tillräcklig konfigurationshantering för testvaran
- Införa lämpliga mätetal för att mäta teststatus och utvärdera kvaliteten på testningen och produkten
- Stödja val och implementering av verktyg till stöd för testprocessen, inklusive budgetrekommendation för val av verktyg (och möjligtvis inköp och/eller support), tilldela tid och arbetsinsats för pilotprojekt och tillhandahålla kontinuerligt stöd under användningen av verktygen
- Besluta om implementering av testmiljöer
- Hjälpa och lyfta fram testarna, testteamet och testyrket inom organisationen
- Utveckla färdigheter och karriärer för testare (till exempel genom utbildningsplaner, prestationsutvärderingar och coaching)

Hur rollen som testledare eller testansvarig genomförs varierar beroende på programvarans utvecklingslivscykel. Vid till exempel agil utveckling hanteras några av de arbetsuppgifter som nämns ovan av det agila teamet. Detta gäller framför allt arbetsuppgifter som rör den dagliga testningen som utförs av teamet, som ofta utförs av en testare som arbetar i teamet. En del av arbetsuppgifterna som spänner över flera team eller hela organisationen, eller som rör personalhantering, kan utföras av

testledare eller testansvariga utanför utvecklingsteamet. Dessa personer kallas ibland testcoacher. Se Black 2009 för mer information om ledning av testprocesserna.

Typiska arbetsuppgifter för testare kan omfatta:

- Granska och bidra till testplaner
- Analysera, granska och utvärdera krav, användarberättelser, acceptanskriterier, specifikationer och modeller för testbarhet (det vill säga testbasen)
- Identifiera och dokumentera testvillkor och underhålla spårbarhet mellan testfall, testvillkor och testbasen
- Designa, organisera och verifiera testmiljöer, ofta tillsammans med systemadministratörer och nätverksansvariga
- Designa och implementera testfall och testprocedurer
- Förbereda och inhämta testdata
- Skapa ett detaljerat testexekveringsschema
- Exekvera tester, utvärdera resultaten och dokumentera avvikelser från de förväntade resultaten
- Använda lämpliga verktyg för att underlätta testprocessen
- Automatisera tester vid behov (kan stödjas av en utvecklare eller expert på testautomatisering)
- Utvärdera icke-funktionella egenskaper som prestandauppfyllelse, tillförlitlighet, användbarhet, säkerhet, kompatibilitet och portabilitet
- Granska tester som har utvecklats av andra

Personer som arbetar med testanalys, testdesign, specifika testtyper eller testautomatisering kan vara specialister i dessa roller. Beroende på produktriskerna, projektriskerna och den modell som valts för programvarans utvecklingslivscykel, kan olika personer ta rollen som testare på olika testnivåer. På komponenttestnivå och komponentintegrationstestnivå sköts till exempel rollen som testare ofta av utvecklarna. Efter acceptanstestnivån hanteras rollen som testare ofta av verksamhetsanalytiker, ämnesexperter och användare. På systemtestnivå och systemintegrationstestnivå sköts rollen som testare ofta av ett oberoende testteam. På driftsacceptanstestnivå sköts rollen som testare ofta av drifts- och/eller systemadministrationspersonal.

5.2 Testplanering och testuppskattning

5.2.1 Testplanens syfte och innehåll

En testplan beskriver testaktiviteterna för utvecklings- och underhållsprojekten. Planeringen påverkas av organisationens testpolicy och teststrategi, de utvecklingslivscykler och metoder som används (se avsnitt 2.1), omfattningen av testning, mål, risker, begränsningar, kriticitet, testbarhet och resurstillgången.

När projektet och testplaneringen framskrider blir mer information tillgänglig och mer detaljer kan tas med i testplanen. Testplanering är en kontinuerlig aktivitet som genomförs under produktens livscykel. (Observera att produktens livscykel kan sträcka sig utanför projektets omfattning och även inkludera underhållsfasen.) Återkoppling från testaktiviteterna ska användas för att upptäcka förändrade risker så att planeringen kan justeras. Planeringen kan dokumenteras i en övergripande testplan och i separata testplaner för testnivåerna, till exempel systemtestning och acceptanstestning, eller för separata testtyper,

till exempel användbarhetstestning och prestandatestning. Testplaneringsaktiviteterna kan omfatta följande, varav en del kan dokumenteras i en testplan:

- Fastställa omfattning, mål och risker för testningen
- Definiera det övergripande testangreppssättet
- Integrera och samordna testaktiviteterna till aktiviteter i programvarans livscykel
- Fatta beslut om vad som ska testas, vilka personer och andra resurser som krävs för de olika testaktiviteterna och hur testaktiviteterna ska genomföras
- Schemalägga aktiviteter för testanalys, design, implementering, exekvering och utvärdering, antingen på särskilda datum (vid till exempel sekventiell utveckling) eller i samband med varje iteration (vid till exempel iterativ utveckling)
- Välja mätetal för testövervakning och teststyrning
- Budgetera för testaktiviteterna
- Fastställa detaljnivå och struktur för testdokumentationen (till exempel genom att tillhandahålla mallar eller exempeldokument)

Testplanernas innehåll varierar och kan sträcka sig utöver vad som anges ovan. Exempel på testplaner finns i ISO-standarden (ISO/IEC/IEEE 29119-3).

5.2.2 Teststrategi och testangreppssätt

En teststrategi ger en generaliserad beskrivning av testprocessen, vanligtvis på produkt- eller organisationsnivå. Vanliga typer av teststrategier omfattar:

- **Analytisk:** Den här typen av teststrategi bygger på en analys av någon faktor (till exempel krav eller risk). Riskbaserad testning är ett exempel på ett analytiskt arbetssätt där tester designas och prioriteras baserat på risknivån.
- **Modellbaserad:** I den här typen av teststrategi baseras testdesignen på en modell av en nödvändig aspekt av produkten, till exempel en funktion, en verksamhetsprocess, en intern struktur eller en icke-funktionell egenskap (till exempel tillförlitlighet). Exempel på sådana modeller omfattar verksamhetsprocessmodeller, tillståndsmodeller och ”reliability growth models”.
- **Metodisk:** Den här typen av teststrategi förlitar sig på systematisk användning av fördefinierade uppsättningar med tester eller testvillkor, till exempel en taxonomi av vanliga eller sannolika typer av felsymptom, en lista över viktiga kvalitetsegenskaper eller företagstäckande standarder för utseende och känsla för mobilappar eller webbsidor.
- **Processanpassad** (eller standardanpassad): Den här typen av teststrategi omfattar analys, design och implementering av tester baserat på externa regler och standarder. Det kan vara sådana som anges av branschspecifika standarder, processdokumentation, i strikt identifiering och användning av testbasen eller genom andra processer och standarder som införs för eller av organisationen.
- **Styrd** (eller konsultativ): Den här typen av teststrategi drivs främst genom råd, vägledning eller instruktioner från intressenter, experter på verksamhetsdomänen eller tekniska experter, som kan finnas utanför testteamet eller utanför själva organisationen.
- **Regressionsfokuserad:** Den här typen av teststrategi motiveras av en önskan att undvika regression av befintlig kapacitet. Teststrategin omfattar återanvändning av befintlig testvara

(särskilt testfall och testdata), omfattande automatisering av regressionstester och standardtestsviter.

- **Reaktiv:** I den här typen av teststrategi är testningen reaktiv i förhållande till komponenten eller systemet som testas och de händelser som inträffar under testkörningen, snarare än att vara planerad i förväg (som i föregående strategier). Testerna designas och implementeras och kan exekveras omedelbart som svar på kunskap som inhämtats från tidigare testresultat. Utforskande testning är en teknik som ofta används i reaktiva strategier.

En lämplig teststrategi skapas ofta genom att kombinera flera av dessa typer av teststrategier. Riskbaserad testning (en analytisk strategi) kan till exempel kombineras med utforskande testning (en reaktiv strategi). De kompletterar varandra och kan ge en effektivare testning när de används tillsammans.

Även om teststrategin ger en allmän beskrivning av testprocessen, skräddarsyr testangreppssättet den aktuella teststrategin för ett visst projekt eller release. Testangreppssättet är startpunkten för valet av testtekniker, testnivåer och testtyper, samt för att definiera startkriterier och avslutskriterier (eller ”definition of ready” respektive ”definition of done”). Strategin skräddarsys genom beslut som fattas i förhållande till projektets komplexitet och mål, vilken typ av produkt som utvecklas och produktriskanalysen. Det valda angreppssättet beror på sammanhanget och kan ta hänsyn till faktorer som risker, säkerhet, tillgängliga resurser och färdigheter, teknik, systemets egenskaper (till exempel specialbyggt jämfört med kommersiellt från hyllan), testmål och bestämmelser.

5.2.3 Startkriterier och avslutskriterier (”Definition of Ready” och ”Definition of Done”)

För att kunna styra programvarans kvalitet och testningen på ett effektivt sätt bör det finnas kriterier som definierar när en given testaktivitet bör starta och när aktiviteten är färdig. Startkriterier (som vanligtvis kallas ”definition of ready” i agil utveckling) definierar förutsättningarna för att genomföra en given testaktivitet. Om startkriterierna inte uppfylls är det sannolikt att aktiviteten blir svårare, mer tidskrävande, dyrare och mer riskfylld. Avslutskriterier (som vanligtvis kallas ”definition of done” i agil utveckling) definierar vilka villkor som måste uppnås för att förklara en testnivå eller en uppsättning tester som slutförda. Start- och avslutskriterier bör definieras för varje testnivå och testtyp och kommer att skilja sig åt beroende på testmål.

Typiska startkriterier omfattar:

- Tillgång till testbara krav, användarberättelser och/eller modeller (till exempel vid strategier för modellbaserad testning)
- Tillgång till testobjekt som uppfyller avslutskriterierna för någon av föregående testnivåer
- Tillgång till testmiljön
- Tillgång till nödvändiga testverktyg
- Tillgång till testdata och andra nödvändiga resurser

Typiska avslutskriterier omfattar:

- Planerade tester har genomförts
- En definierad täckningsgrad (till exempel för krav, användarberättelser, acceptanskriterier, risker eller kod) har uppnåtts
- Antalet olösta defekter ligger inom en överenskommen gräns

- Antalet beräknade kvarvarande defekter är tillräckligt lågt
- De utvärderade nivåerna av tillförlitlighet, prestandauppfyllelse, användbarhet, informationssäkerhet och andra relevanta kvalitetsegenskaper är tillräckliga

Även om inte avslutskriterierna är uppfyllda är det vanligt att testaktiviteterna förkortas på grund av att budgeten är förbrukad, den schemalagda tiden har löpt ut och/eller press att få ut produkten på marknaden. Det kan vara acceptabelt att avsluta testningen under sådana förhållanden om projektets intressenter och verksamhetsägarna har granskat och accepterat riskerna med att ta systemet i drift utan mer testning.

5.2.4 Testexekveringsschema

När de olika testfallen och testprocedurerna har producerats (där en del testprocedurer eventuellt kan vara automatiserade) och ordnats i testsviter, kan testsviterna ordnas i ett testexekveringsschema som definierar i vilken ordning de ska köras. Testexekveringsschemat bör ta hänsyn till faktorer som prioritering, beroenden, omtestning, regressionstester och den mest effektiva ordningen för att exekvera testerna.

Helst bör testfallen ordnas så att de körs i prioritetsordning, vanligtvis genom att köra testfallen som har högst prioritet först. Det här arbetssättet kanske inte fungerar om testfallen har beroenden eller om de funktioner som testas har beroenden. Om ett testfall med högre prioritet är beroende av ett testfall med lägre prioritet, måste testfallet med lägre prioritet exekveras först. På samma sätt gäller att om det finns beroenden som sträcker sig över flera testfall, måste dessa ordnas på lämpligt sätt oavsett relativa prioriteringar. Omtestning och regressionstester måste också prioriteras baserat på hur viktigt det är med snabb återkoppling efter ändringar, men även här kan det finnas beroenden.

I vissa fall är olika testsekvenser möjliga, med olika nivåer av verkningsgrad förknippade med de olika sekvenserna. I så fall måste kompromisser göras mellan uppfyllelsen av testkörningen och att följa prioriteringen.

5.2.5 Faktorer som påverkar testarbetet

Uppskattning av testarbetsmängden omfattar att förutse mängden testrelaterat arbete som krävs för att uppfylla testningens målsättningar för ett visst projekt, en viss release eller en viss iteration. Faktorer som påverkar testarbetet kan omfatta produktens egenskaper, utvecklingsprocessens egenskaper, personernas egenskaper och testresultaten, enligt nedan.

Produktegenskaper

- Riskerna som är förknippade med produkten
- Testbasens kvalitet
- Produktens storlek
- Produktdomänens komplexitet
- Kraven på kvalitetsegenskaperna (till exempel informationssäkerhet och tillförlitlighet)
- Den detaljnivå som krävs för testdokumentationen
- Kraven på rättslig och reglerande efterlevnad

Utvecklingsprocessens egenskaper

- Organisationens stabilitet och mognad

- Utvecklingsmodellen som används
- Angreppssätt för test
- Verktyg som används
- Testprocessen
- Tidspress

Personegenskaper

- Färdigheter och erfarenhet hos inblandade personer, särskilt när det gäller liknande projekt och produkter (till exempel domänkunskap)
- Teamets sammanhållning och ledarskap

Testresultat

- Antal och allvarlighetsgrad för upptäckta defekter
- Mängden omarbeten som krävs

5.2.6 Testuppskattningstekniker

Det finns ett antal uppskattningstekniker som används för att fastställa det arbete som krävs för tillräcklig testning. Två av de vanligaste teknikerna är:

- Mätetalsbaserad teknik: beräkna testarbetet baserat på mätetalen från tidigare liknande projekt, eller baserat på typiska värden
- Expertbaserad teknik: uppskatta testarbetet baserat på erfarenheten hos ägarna till testuppgifterna eller av experter

Vid agil utveckling är till exempel ”burndown charts” exempel på det mätetalsbaserade arbetssättet eftersom arbetet registreras, rapporteras och sedan vägs mot teamets arbetshastighet för att avgöra hur mycket arbete teamet kan göra i nästa iteration. ”Planning poker” är ett exempel på det expertbaserade arbetssättet, eftersom teammedlemmarna beräknar det arbete som krävs för att leverera en funktion baserat på sin erfarenhet (ISTQB-AT Foundation Level Agile Tester Extension Syllabus).

I sekventiella projekt är modeller för att åtgärda defekter exempel på det mätetalsbaserade arbetssättet, eftersom volymer av defekter och den tid som det tar att åtgärda dem registreras och rapporteras för att utgöra en bas för uppskattning av kommande projekt av liknande typ. Wideband Delphi-uppskattningstekniken är å andra sidan ett exempel på det expertbaserade arbetssättet där grupper av experter tillhandahåller uppskattningar som bygger på deras erfarenhet (ISTQB-ATM Advanced Level Test Manager Syllabus).

5.3 Testövervakning och teststyrning

Syftet med testövervakning är att samla information och ge återkoppling samt skapa synlighet för testaktiviteter. Information som ska övervakas kan samlas in manuellt eller automatiskt och bör användas för att uppskatta teststatusen samt mäta om testavslutskriterier, eller de testarbetsuppgifter som förknippas med ett agilt projekts ”definition of done”, är uppfyllda. Exempel är om de uppfyller målen avseende täckningsgraden för produktrisker, krav eller acceptanskriterier.

Teststyrning beskriver all vägledning eller alla korrigerande åtgärder som vidtagits som följd av information och mätetal som samlats in och (eventuellt) rapporterats. Åtgärderna kan täcka alla testaktiviteter och kan påverka all annan aktivitet i livscykeln för programvaran.

Exempel på teststyrningsåtgärder omfattar:

- Omprioritera tester när en identifierad risk inträffar (till exempel programvara som levereras sent)
- Ändra testschemat med anledning av om en testmiljö eller annan resurs är tillgänglig eller inte
- Omvärdera om ett testobjekt uppfyller ett start- eller avslutskriterium med anledning av omarbete

5.3.1 Mätetal som används i testning

Mätetal kan samlas in under och i slutet av testaktiviteterna för att utvärdera:

- Framsteg jämfört med planerat schema och budget
- Testobjektets aktuella kvalitet
- Testangreppssättets lämplighet
- Effektiviteten hos testaktiviteterna i förhållande till målen

Vanliga mätetal för tester omfattar:

- Procentuell andel av planerat arbete under förberedelsen av testfallen (eller procentuell andel av planerade testfall som implementerats)
- Procentuell andel av planerat arbete som utförts under förberedelsen av testmiljön
- Testfallsexekvering (till exempel antal testfall som körts/inte körts, testfall som godkänts/underkänts och/eller testvillkor som godkänts/underkänts)
- Defektinformation (till exempel feltäthet, defekter som upptäckts och åtgärdats, felfrekvens och resultat av omtestning)
- Testtäckning av krav, användarberättelser, acceptanskriterier, risker eller kod
- Avslutade arbetsuppgifter, resurstilldelning, resursanvändning och arbetsinsats
- Testkostnad, inklusive kostnaden jämfört med fördelarna med att hitta nästa defekt eller kostnaden jämfört med fördelen av att köra nästa test

5.3.2 Syfte, innehåll och målgrupper för testrapporter

Syftet med testrapportering är att sammanfatta och informera om testaktiviteter, både under och i slutet av testaktiviteten (till exempel en testnivå). En testrapport som förbereds under en testaktivitet kallas teststatusrapport, medan en testrapport som förbereds i slutet av en testaktivitet kallas sammanfattande testrapport.

Under testövervakning och teststyrning lämnar testledaren regelbundet teststatusrapporter till intressenterna. Förutom innehåll som är gemensamt för teststatusrapporter och sammanfattande testrapporter, kan typiska teststatusrapporter också innehålla:

- Statusen för testaktiviteterna och framsteg gentemot testplanen
- Faktorer som hindrar framsteg
- Testning planerad för nästa rapportperiod
- Testobjektets kvalitet

När avslutskriterierna uppnåtts sänder testledaren ut en sammanfattande testrapport. Den innehåller en sammanfattning av de genomförda testerna, baserat på den senaste teststatusrapporten och annan relevant information.

Typiska teststatusrapporter och sammanfattande testrapporter kan omfatta:

- Sammanfattning av genomförd testning
- Information om vad som inträffat under en testperiod
- Avvikelser från planen, inklusive avvikelser i tidsplaneringen, varaktighet eller arbetsmängd för testaktiviteterna
- Testningens status och produktkvalitet med avseende på avslutskriterier eller ”definition of done”
- Faktorer som har blockerat eller fortsätter att blockera framstegen
- Mätetal för defekter, testfall, testtäckning, aktiviteternas status och resursförbrukning. (till exempel enligt beskrivningen i 5.3.1)
- Återstående risker (se avsnitt 5.5)
- Producerade testarbetsprodukter som kan återanvändas

Innehållet i en testrapport kan variera beroende på projektet, organisatoriska krav samt programvarans utvecklingslivscykel. Ett komplext projekt med många intressenter eller ett styrt projekt kan till exempel kräva mer detaljerad och strikt rapportering än en snabb programvaruuppdatering. Ett annat exempel är att teststatusrapportering vid agil utveckling kan tas med i aktivitetstavlor, defektsammanfattningar och ”burndown charts”, som kan diskuteras under dagliga stående möten (se ISTQB-AT Foundation Level Agile Tester Extension Syllabus).

Förutom att anpassa testrapporterna baserat på projektets sammanhang, bör testrapporter skräddarsys baserat på rapportens målgrupp. Typ och mängd av information som bör inkluderas för en teknisk målgrupp eller ett testteam kan skilja sig från vad som skulle tas med i en sammanfattande ledningsrapport. I det första fallet kan detaljerad information om typer av defekter och trender vara viktig. I det senare fallet kan en rapport på hög nivå (till exempel en statussammanfattning av defekter enligt prioritet, budget, tidsplan och testvillkor som blivit godkända/underkända/inte testats) vara mer lämplig.

ISO-standarden (ISO/IEC/IEEE 29119-3) hänvisar till två typer av testrapporter, teststatusrapporter och testavslutsrapporter (kallas sammanfattande testrapporter i den här kursplanen), och innehåller strukturer och exempel för varje typ.

5.4 Konfigurationshantering

Syftet med konfigurationshantering är att etablera och bibehålla integriteten hos komponenten eller systemet, testvaran, och deras relationer med varandra under projektet och produktens livscykel.

För att stödja testningen på rätt sätt bör konfigurationshanteringen omfatta att säkerställa följande:

- Alla testelement är unikt identifierade, versionskontrollerade, uppföljda efter förändringarna och relaterade till varandra
- Alla element i testvaran är unikt identifierade, versionskontrollerade, uppföljda efter förändringarna, relaterade till varandra och relaterade till versionerna av testelementen så att spårbarheten kan bibehållas under testprocessen
- Alla identifierade dokument och programvaruelement är otvetydigt definierade i testdokumentationen

Under testplaneringen ska procedurer för konfigurationshantering och infrastruktur (verktyg) identifieras och implementeras.

5.5 Risker och testning

5.5.1 Definition av risker

Risker omfattar möjligheten av en händelse i framtiden som har negativa konsekvenser. Risknivån fastställs genom sannolikheten och påverkan (skadan) av händelsen.

5.5.2 Produkt- och projektrisker

Produktrisker omfattar sannolikheten att en arbetsprodukt (till exempel en specifikation, en komponent, ett system eller ett test) inte uppfyller de legitima behoven hos användare och/eller intressenter. När produktrisker förknippas med specifika kvalitetsegenskaper för en produkt (till exempel funktionell lämplighet, tillförlitlighet, prestandauppfyllelse, användbarhet, säkerhet, kompatibilitet, underhållbarhet och portabilitet), kallas produktrisker även kvalitetsrisker. Exempel på produktrisker omfattar:

- Programvaran kanske inte utför sina avsedda funktioner i enlighet med specifikationen
- Programvaran kanske inte utför sina avsedda funktioner i enlighet med användarens, kundens och/eller intressenternas behov
- En systemarkitektur kanske inte ger tillräckligt stöd för en del icke-funktionella krav
- En viss beräkning kan genomföras på felaktigt sätt under vissa förutsättningar
- En styrstruktur för en loop kan vara felaktigt kodad
- Svarstiderna kan vara otillräckliga för ett högpresterande system för behandling av transaktioner
- Återkoppling av användarupplevelsen (UX) kanske inte uppfyller produktförväntningarna

Projektriskerna omfattar situationer som om de skulle uppstå kan ha negativ effekt på projektets förmåga att uppfylla målen. Exempel på projektrisker omfattar:

- Projektproblem:
    - Förseningar kan uppstå under leverans, uppgiftens slutförande eller uppfyllandet av avslutskriterier eller ”definition of done”
    - Felaktiga uppskattningar, omfördelning av resurser till projekt med högre prioritet eller allmänna kostnadsbesparingar inom organisationen kan leda till otillräcklig finansiering
    - Sena ändringar kan leda till betydande omarbetning
- Organisatoriska problem:
    - Färdigheter, utbildning och tillgången till personal kanske inte är tillräcklig
    - Personalärenden kan orsaka konflikter och andra problem
    - Användare, verksamhetspersonal eller ämnesexperter kanske inte finns tillgängliga på grund av konkurrerande verksamhetsprioriteringar
- Politiska problem:
    - Testarna kanske inte förmedlar sina behov och/eller testresultaten tillräckligt bra

  - Utvecklare och/eller testare kanske inte följer upp information som kommer fram under testning och granskningar (till exempel att inte förbättrar arbetssätten för utveckling och test)
  - Det kan råda en olämplig attityd mot eller olämpliga förväntningar på testningen (till exempel bristande uppskattning av värdet av att hitta defekter under testningen)
- Tekniska problem:
  - Kraven kanske inte är tillräckligt väl definierade
  - Kraven kanske inte uppfylls på grund av befintliga begränsningar
  - Testmiljön kanske inte är färdig i tid
  - Datakonvertering, migrationsplanering och tillhörande verktyg kan vara försenade
  - Brister i utvecklingsprocessen kan påverka konsekvensen eller kvaliteten hos ett projekt, till exempel i form av design, kod, konfiguration, testdata och testfall
  - Dålig felhantering och liknande problem kan resultera i ackumulerade defekter och andra tekniska brister
- Leverantörsproblem:
  - En tredje part kanske inte levererar en nödvändig produkt eller tjänst, eller går i konkurs
  - Kontraktsmässiga problem kan orsaka problem för projektet

Projektrisker kan påverka både utvecklingsaktiviteter och testaktiviteter. I vissa fall kan projektledarna vara ansvariga för att hantera projektriskerna, men det är inte ovanligt att testledarna har ansvar för testrelaterade projektrisker.

5.5.3 Riskbaserad testning och produktkvalitet

Risker används för att fokusera på det arbete som krävs under testningen. De används för att avgöra var och när testningen ska påbörjas och identifiera områden som behöver mer uppmärksamhet. Testning används för att minska sannolikheten för att en negativ effekt ska inträffa, eller för att minska påverkan av en negativ effekt. Testning används som en riskreducerande aktivitet, för att ge återkoppling om identifierade risker samt för att lämna återkoppling om kvarvarande (olösta) risker.

En riskbaserad inställning till testning ger proaktiva möjligheter att minska produktrisknivåerna. Den innefattar en produktriskanalys som inkluderar identifiering av produktriskerna och en utvärdering av sannolikhet och påverkan för varje risk. Den resulterande produktriskinformationen används för att vägleda testplaneringen, specificering, förberedelser och exekvering av testfallen, samt testövervakning och teststyrning. Att analysera produktriskerna tidigt bidrar till att göra projektet framgångsrikt.

Med ett riskbaserat arbetssätt kan resultaten av produktriskanalysen användas för att:

- Fastställa vilka testtekniker som ska användas
- Fastställa specifika nivåer och typer av testning som ska användas (till exempel säkerhetstestning eller tillgänglighetstestning)
- Fastställa i vilken omfattning testningen ska genomföras
- Prioritera testningen i syfte att hitta kritiska defekter så tidigt som möjligt
- Fastställa om några aktiviteter utöver testning kan användas för att minska riskerna (till exempel att tillhandahålla utbildning till oerfarna designers)

Riskbaserad testning utnyttjar den kollektiva kunskapen och insikterna hos projektets intressenter för att genomföra en produktriskanalys. För att säkerställa att risken för felsymptom i produkten minimeras, utgör riskhanteringsaktiviteterna ett disciplinerat arbetssätt för att:

- Analysera (och regelbundet göra nya utvärderingar av) vad som kan gå fel (risker)
- Fastställa vilka risker som är viktiga att hantera
- Införa åtgärder för att motverka dessa risker
- Ta fram beredskapsplaner för att hantera riskerna om de skulle bli verkliga händelser

Dessutom kan testningen identifiera nya risker, bidra till att fastställa vilka risker som ska reduceras och minska osäkerheten kring riskerna.

5.6 Felhantering

Eftersom ett av målen med testningen är att hitta defekter, bör defekter som upptäcks under testningen loggas. Hur defekterna loggas kan variera beroende på sammanhanget för komponenten eller systemet som testas, testnivån och modellen för programvarans utvecklingslivscykel. Alla defekter som identifieras bör undersökas och spåras från upptäckt och klassificering till att de är lösta (till exempel åtgärda defekter och bekräfta att de är lösta genom omtestning, skjuta upp dem till en kommande release, acceptera dem som en permanent produktbegränsning o.s.v.). För att hantera alla defekter tills de är lösta bör en organisation etablera en process för felhantering som omfattar ett arbetsflöde och regler för klassificering. Alla som deltar i felhanteringen, inklusive designers, utvecklare, testare och produktägare måste vara överens om processen. Inom vissa organisationer kan loggning och spårning av defekter ske mycket informellt.

Under felhanteringsprocessen kan vissa av rapporterna visa sig beskriva falska positiva resultat, inte verkliga felsymptom som beror på defekter. Ett test kan till exempel bli underkänt om en nätverksanslutning bryts eller tidsgränsen passeras. Det här beteendet uppstår inte genom en defekt i testobjektet, utan är en anomali som behöver undersökas. Testarna bör försöka minimera antalet falska positiva resultat som rapporteras som defekter.

Defekter kan rapporteras under kodning, statisk analys, granskningar, dynamisk testning eller användning av en programvaruprodukt. Defekter kan rapporteras för problem i koden eller för system, eller i någon typ av dokumentation inklusive krav, användarberättelser och acceptanskriterier, utvecklingsdokument, testdokument, användarmanualer eller installationshandledningar. För att skapa en effektiv process för felhantering kan organisationer skapa standarder för attribut, klassificering och arbetsflöde för defekter.

Typiska felrapporter har följande mål:

- Förse utvecklare och andra med information om bieffekter som inträffat, ge dem möjlighet att identifiera specifika effekter, isolera problemet med minimala reproduceringstester samt korrigera potentiella defekter, enligt behov eller för att i övrigt lösa problemet
- Förse testledare med ett sätt att följa upp arbetsproduktens kvalitet och inverkan på testningen (om till exempel många defekter rapporteras måste testarna lägga mycket tid på att rapportera dem i stället för att köra tester, och det kommer att behövas mer omtestning)
- Ge idéer för förbättring av utveckling och testprocesser

En felrapport som skrivits under dynamisk testning omfattar normalt sett:

- En identifierare
- En titel och en kort sammanfattning av den defekt som rapporteras

- Felrapportens datum, utfärdande organisation och författare
- Identifiering av testelement (konfigurationsobjektet som testas) och miljön
- Den fas under utvecklingslivscykeln där defekten observerades
- En beskrivning av defekten för att möjliggöra återskapande och lösning, inklusive loggar, skärmbilder av databasdumpar eller inspelningar (om de upptäcks under testexekvering)
- Förväntat och faktiskt resultat
- Defektens omfattning eller påverkan (allvarlighet) sett ur intressenternas perspektiv
- Vikt/prioritet att korrigera
- Felrapportens status (till exempel öppen, uppskjuten (deferred), dubblett, väntar på korrigering, väntar på omtest, återöppnad eller stängd)
- Slutsatser, rekommendationer och godkännanden
- Globala problem, till exempel andra områden som kan påverkas av en förändring som uppstått genom defekten
- Ändringshistorik, till exempel sekvens av åtgärder som vidtas av projektteamets medlemmar i syfte att isolera, reparera och bekräfta defekten som korrigerad
- Referenser, inklusive det testfall som avslöjade problemet

En del av den här informationen kan inkluderas och/eller hanteras automatiskt vid användning av felhanteringsverktyg, till exempel automatisk tilldelning av en identifierare eller tilldelning och uppdatering av felrapportens status under arbetsflödet. Defekter som upptäcks under statisk testning, och framför allt vid granskningar, kommer normalt sett att dokumenteras på annat sätt, till exempel i anteckningar från granskningsmötet.

Ett exempel på innehållet i en felrapport finns i ISO-standarden (ISO/IEC/IEEE 29119-3) (som hänvisar till felrapporter som avvikelserapporter).

6 Verktygsstöd för testning — 40 minuter

Nyckelord

datadriven testning, nyckelordsdriven testning, prestandatestverktyg, testautomatisering, testexekveringsverktyg, testledningsverktyg

Utbildningsmål för testverktyg

6.1 Överväganden för testverktyg

FL-6.1.1 (K2) Klassificera testverktyg i enlighet med deras syfte och de testaktiviteter som de stöder

FL-6.1.2 (K1) Identifiera fördelar och risker med testautomatisering

FL-6.1.3 (K1) Komma ihåg särskilda överväganden för testexekvering och testledningsverktyg

6.2 Effektiv användning av verktyg

FL-6.2.1 (K1) Identifiera huvudprinciperna för val av verktyg

FL-6.2.2 (K1) Komma ihåg målen med pilotprojekt för att introducera verktyg

FL-6.2.3 (K1) Identifiera framgångsfaktorer för utvärdering, implementering, införande och kontinuerlig support av testverktyg i en organisation

6.1 Överväganden för testverktyg

Testverktyg kan användas för att stödja en eller flera testaktiviteter. Sådana verktyg omfattar:

- Verktyg som används direkt i testningen, till exempel textexekveringsverktyg och verktyg för förberedelse av testdata
- Verktyg som hjälper till att hantera krav, testfall, testprocedurer, automatiserade testskript, testresultat, testdata och defekter, samt används för att rapportera och övervaka testexekvering
- Verktyg som används för utredning och utvärdering
- Alla verktyg som bidrar till testningen (ett kalkylblad är också ett testverktyg i det här avseendet)

6.1.1 Klassificering av testverktyg

Testverktyg kan ha ett eller flera syften beroende på sammanhanget:

- Förbättra testaktiviteternas effektivitet genom att automatisera repetitiva uppgifter eller uppgifter som kräver betydande resurser när de utförs manuellt (till exempel testexekvering, regressionstestning)
- Förbättra effektiviteten hos testaktiviteterna genom att stödja manuella testaktiviteter under hela testprocessen (se avsnitt 1.4)
- Förbättra kvaliteten på testaktiviteterna genom att tillåta mer konsekvent testning och en högre nivå av reproducerbarhet
- Automatisera aktiviteter som inte går att göra manuellt (till exempel prestandatestning i stor skala)
- Öka tillförlitligheten i testningen (till exempel genom att automatisera stora datajämförelser eller simulera beteende)

Verktyg kan klassificeras baserat på flera kriterier, till exempel syfte, pris, licensmodell (kommersiell eller open source) och den teknik som används. I den här kursplanen klassificeras verktygen enligt de testaktiviteter de stödjer.

En del verktyg stödjer tydligt enbart eller huvudsakligen en aktivitet, medan andra kan stödja fler. Verktygen klassificeras dock efter den aktivitet som de är närmast förknippade med. Verktyg från en enda leverantör, särskilt de som har designats för att användas tillsammans, kan levereras som en enda integrerad svit.

En del typer av testverktyg kan vara inkräktande, vilket innebär att de kan påverka själva testutfallet. Till exempel kan de verkliga svarstiderna för en applikation variera på grund av de extra instruktioner som exekveras med ett prestandatestverktyg, eller så kan resultatet från den kodtäckning som uppnås förvrängas på grund av användningen av ett verktyg för mätning av täckningsgraden. Konsekvensen av att använda inkräktande verktyg kallas för probeffekten.

En del verktyg ger stöd som normalt sett är mer lämpligt för utvecklare (till exempel verktyg som används under komponent- och integrationstestning). Sådana verktyg markeras med ”(U)” i avsnitten nedan.

Verktygsstöd för hantering av testning och testvara

Testhanteringsverktyg kan användas för samtliga testaktiviteter under hela programvarans utvecklingslivscykel. Exempel på verktyg som stödjer hantering av testning och testvara:

- Testhanteringsverktyg och verktyg för applikationslivscykelhantering (ALM)
- Kravhanteringsverktyg (till exempel spårbarhet till testobjekt)

- Felhanteringsverktyg
- Konfigurationshanteringsverktyg
- Verktyg för kontinuerlig integration (U)

Verktygsstöd för statisk testning

Verktyg för statisk testning är förknippade med de aktiviteter och fördelar som beskrivs i kapitel 3. Exempel på sådana verktyg är:

- Verktyg som stödjer granskningar
- Verktyg för statisk analys (U)

Verktygsstöd för testdesign och testimplementation

Testdesignverktyg underlättar skapandet av underhållbara arbetsprodukter vid testdesign och implementation, inklusive testfall, testprocedurer och testdata. Exempel på sådana verktyg är:

- Testdesignverktyg
- Verktyg för modellbaserad testning
- Verktyg för att förbereda testdata
- Verktyg för acceptanstestdriven utveckling (ATDD) och beteendedriven utveckling (BDD)
- Verktyg för testdriven utveckling (TDD) (U)

I en del fall kan verktyg som stödjer testdesign och implementation också ha stöd för testexekvering och loggning.

Verktygsstöd för testexekvering och loggning

Det finns många verktyg som stödjer och förbättrar testexekvering och loggning. Exempel på sådana verktyg är:

- Testexekveringsverktyg (till exempel för körning av regressionstester)
- Verktyg för test av täckningsgrad (till exempel kravtäckning och kodtäckning (U))
- Testexekveringsplattformar (U)
- Verktyg för enhetstestramverk (U)

Verktygsstöd för prestandamätning och dynamisk analys

Verktyg för prestandamätning och dynamisk analys är avgörande för att stödja aktiviteter för prestanda- och lasttestning, eftersom dessa aktiviteter inte kan skötas manuellt. Exempel på sådana verktyg är:

- Prestandatestverktyg
- Övervakningsverktyg
- Dynamiskt analysverktyg (U)

Verktygsstöd för specialiserade testbehov

Förutom verktyg som stödjer den allmänna testprocessen, finns det många andra verktyg som stödjer mer specifika testproblem. Exempel på sådana är verktyg som fokuserar på:

- Utvärdering av datakvalitet

- Datakonvertering och migrering
- Användbarhetstestning
- Tillgänglighetstestning
- Lokaliseringstestning
- Informationssäkerhetstestning
- Portabilitetstestning (till exempel test av programvara över flera plattformar som stöds)

6.1.2 Fördelar och risker med testautomatisering

Att bara skaffa ett verktyg är ingen garanti för att lyckas. Varje nytt verktyg som införs i en organisation kräver arbete för att uppnå verkliga och varaktiga fördelar. Det finns potentiella fördelar och möjligheter med att använda verktyg i testning, men det finns också risker. Detta gäller framför allt för testexekveringsverktyg (som ofta kallas för testautomatisering).

Potentiella fördelar med att använda verktyg till stöd för testexekvering omfattar:

- Minskning av repetitivt manuellt arbete (till exempel körning av regressionstester, arbetsuppgifter för att sätta upp och ta ned testmiljöer, mata in samma testdata på nytt samt att kontrollera mot kodningsstandarder), vilket sparar tid
- Större konsistens och repeterbarhet (till exempel att testdata skapas på ett sammanhängande sätt, tester körs med ett verktyg i samma ordning och med samma frekvens samt att tester hela tiden härleds från kraven)
- Mer objektiv utvärdering (till exempel statiska mätningar och täckningsgrad)
- Enklare åtkomst till information om testning (till exempel statistik och diagram med teststatus, andel defekter och prestanda)

Potentiella risker med att använda verktyg till stöd för testningen omfattar:

- Förväntningarna på verktyget kan vara orealistiska (inklusive funktioner och användarvänlighet)
- Tid, kostnad och arbete för det första införandet av ett verktyg kan underskattas (inklusive utbildning och extern expertis)
- Den tid och det arbete som krävs för att uppnå stora och varaktiga fördelar av verktyget kan underskattas (inklusive behovet av förändringar av testprocessen och ständiga förbättringar av verktygets användning)
- Det arbete som krävs för att bibehålla de testtillgångar som genereras av verktyget kan underskattas
- För stor vikt kan läggas vid verktyget (som ersättning för testdesign eller exekvering, eller användning av automatiserad testning där manuell testning hade passat bättre)
- Versionshanteringen av testtillgångarna kan försummas
- Problem med relationer och interoperabilitet mellan kritiska verktyg kan försummas, till exempel kravhanteringsverktyg, konfigurationshanteringsverktyg, felhanteringsverktyg och verktyg från flera leverantörer
- Verktygsleverantören kan gå i konkurs, sluta stödja verktyget eller sälja verktyget till en annan leverantör

- Leverantören kan sköta support, uppgraderingar och felrättningar långsamt eller på ett dåligt sätt
- Ett open source-projekt kan upphöra
- En ny plattform eller teknik kanske inte stöds av verktyget
- Det finns kanske ingen tydlig ägare till verktyget (till exempel för mentorskap eller uppdateringar)

6.1.3 Särskilda överväganden för testexekverings- och testhanteringsverktyg

För att få en smidig och lyckad implementering finns det en rad saker som bör övervägas vid val och integration av verktyg för testexekvering och testhantering i en organisation.

Testexekveringsverktyg

Testexekveringsverktyg exekverar testobjekt med hjälp av automatiserade testskript. Den här typen av verktyg kräver ofta stora insatser för att uppnå betydande fördelar.

Att spela in tester genom att registrera åtgärder från en manuell testare verkar attraktivt, men den här metoden är inte skalbar för ett stort antal testskript. Ett inspelat skript är en linjär återgivning med specifika data och åtgärder som del av varje skript. Den här typen av skript kan vara instabilt när oväntade händelser inträffar. Den senaste generationen av dessa verktyg som utnyttjar ”smart” bildinspelningsteknik har ökat användbarheten för verktygen i den här klassen, även om de genererade skripten fortfarande kräver ständigt underhåll när systemets användargränssnitt utvecklas över tiden.

Ett datadrivet angreppssätt för test skiljer ut indata och förväntade resultat, vanligtvis i ett kalkylblad, och använder ett mer generiskt testskript som kan läsa indata och köra samma testskript med olika data. Testare som inte är bekanta med skriptspråket kan därefter skapa nya testdata för dessa fördefinierade skript.

Vid nyckelordsdrivet angreppssätt för test behandlar ett generiskt skript olika nyckelord som beskriver vilka åtgärder som ska vidtas, och dessa anropar sedan nyckelordsskript som behandlar tillhörande testdata. Testare (även om de inte är bekanta med skriptspråket) kan därefter definiera tester med hjälp av nyckelorden och tillhörande data, som kan skräddarsys efter den applikation som testas. Ytterligare information om arbetssätt för datadriven och nyckelordsdriven testning finns i ISTQB-TAE Advanced Level Test Automation Engineer Syllabus, Fewster 1999 och Buwalda 2001.

Ovanstående arbetssätt kräver att det finns en person som har expertkunskaper i skriftspråket (testare, utvecklare eller specialister på testautomatisering). Oavsett vilken skriptteknik som används måste de förväntade resultaten för varje test jämföras med de verkliga resultaten av testet, antingen dynamiskt (medan testet körs) eller sparas till senare jämförelse (efter exekvering).

Verktyg för modellbaserad testning (MBT) spelar in en funktionsspecifikation i form av en modell, till exempel ett aktivitetsdiagram. Den här arbetsuppgiften utförs i allmänhet av en systemdesigner. MBT-verktyget tolkar modellen för att skapa testfallsspecifikationer som sedan kan sparas i testhanteringsverktyget och/eller köras med ett testexekveringsverktyg (se ISTQB-MBT Foundation Level Model-Based Testing Syllabus).

Testhanteringsverktyg

Testhanteringsverktygen behöver ofta samverka med andra verktyg eller kalkylblad av olika anledningar, inklusive att:

- Producera användbar information i ett format som passar organisationens behov
- Bibehålla konsistent spårbarhet enligt kraven i ett kravhanteringsverktyg
- Länka till testobjektets versionsinformation i konfigurationshanteringsverktyget

Detta är särskilt viktigt att tänka på vid användning av ett integrerat verktyg (till exempel livscykelhantering för applikationen, ALM), vilket inkluderar en testhanteringsmodul (och möjligen ett felhanteringssystem) samt andra moduler (till exempel projekttidsplan och budgetinformation) som används av olika grupper i en organisation.

6.2 Effektiv användning av verktyg

6.2.1 Huvudprinciper för val av verktyg

De viktigaste övervägandena vid val av ett verktyg för en organisation omfattar:

- Utvärdering av organisationens mognad samt dess styrkor och svagheter
- Identifiering av möjligheter till en förbättrad testprocess med stöd av verktyg
- Förståelse för de tekniker som används av testobjekten för att välja ett verktyg som är kompatibelt med den tekniken
- Version och verktyg för kontinuerlig integration som redan används i organisationen, i syfte att säkerställa verktygets kompatibilitet och integration
- Utvärdering av verktyget mot tydliga krav och objektiva kriterier
- Övervägande av om verktyget finns tillgängligt för en kostnadsfri provperiod (och hur länge)
- Utvärdering av leverantören (inklusive utbildning, support och kommersiella aspekter) eller support för icke-kommersiella verktyg (open source-verktyg)
- Identifiering av interna krav för handledning och mentorskap vid användning av verktyget
- Utvärdering av utbildningsbehov, med hänsyn till färdigheterna i testning (och testautomatisering) hos de personer som ska arbeta direkt med verktygen
- Övervägande av för- och nackdelar med olika licensmodeller (till exempel kommersiella eller open source)
- Beräkning av förhållandet mellan kostnader och fördelar baserat på ett konkret affärsfall (vid behov)

Som sista steg bör en ”proof-of-concept”-utvärdering genomföras för att fastställa om verktyget presterar effektivt med programvaran som testas och inom den aktuella infrastrukturen, eller vid behov för att identifiera nödvändiga förändringar av infrastrukturen så att verktyget kan användas på ett effektivt sätt.

6.2.2 Pilotprojekt för att införa ett verktyg i en organisation

Efter att ha slutfört valet av verktyg och ett framgångsrikt ”proof-of-concept”, börjar införandet av det valda verktyget i en organisation vanligtvis med ett pilotprojekt som har följande mål:

- Skaffa djupgående kunskap om verktyget och förstå både dess styrkor och svagheter
- Utvärdera hur verktyget passar in med befintliga processer och arbetssätt och fastställa vad som behöver ändras
- Besluta om standardmetoder för att använda, hantera, lagra och bibehålla verktyget och testtillgångarna (till exempel besluta om regler för namngivning av filer och tester, val av kodstandarder, skapande av bibliotek och definition av testsviternas modularitet)
- Bedöma om fördelarna går att uppnå till rimliga kostnader

- Förstå mätetalen som du vill att verktyget ska samla in och rapportera, och konfigurera verktyget för att säkerställa att mätetalen kan spelas in och rapporteras

6.2.3 Framgångsfaktorer för verktyg

Framgångsfaktorer för utvärdering, implementation, införande och pågående stöd för verktyg inom en organisation omfattar att:

- Införa verktyget i resten av organisationen stegvis
- Anpassa och förbättra processerna så att de passar för användningen av verktyget
- Tillhandahålla utbildning, handledning och mentorskap för verktygets användare
- Definiera riktlinjer för användning av verktyget (till exempel interna standarder för automatisering)
- Implementera ett sätt att samla användningsinformation från själva användningen av verktyget
- Övervaka verktygets användning och fördelar
- Tillhandahålla stöd för användarna av ett visst verktyg
- Samla in lärdomar från alla användare

Det är också viktigt att se till att verktyget är tekniskt och organisatoriskt integrerat i programvarans utvecklingslivscykel, vilket kan omfatta separata organisationer ansvariga för driften och/eller separata tredjepartsleverantörer.

Se Graham 2012 för erfarenheter och råd kring användning av testexekveringsverktyg.

7 Referenser

Standarder

ISO/IEC/IEEE 29119-1 (2013) Software and systems engineering – Software testing – Part 1: Concepts and definitions

ISO/IEC/IEEE 29119-2 (2013) Software and systems engineering – Software testing – Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2013) Software and systems engineering – Software testing – Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2015) Software and systems engineering – Software testing – Part 4: Test techniques

ISO/IEC 25010, (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models

ISO/IEC 20246: (2017) Software and systems engineering — Work product reviews

UML 2.5, Unified Modeling Language Reference Manual, http://www.omg.org/spec/UML/2.5.1/, 2017

ISTQB-dokument

ISTQB:s ordlista

ISTQB Foundation Level Overview 2018

ISTQB-MBT Foundation Level Model-Based Tester Extension Syllabus

ISTQB-AT Foundation Level Agile Tester Extension Syllabus

ISTQB-ATA Advanced Level Test Analyst Syllabus

ISTQB-ATM Advanced Level Test Manager Syllabus

ISTQB-SEC Advanced Level Security Tester Syllabus

ISTQB-TAE Advanced Level Test Automation Engineer Syllabus

ISTQB-ETM Expert Level Test Management Syllabus

ISTQB-EITP Expert Level Improving the Test Process Syllabus

Böcker och artiklar

Beizer, B. (1990*) Software Testing Techniques (2e)*, Van Nostrand Reinhold: Boston, MA

Black, R. (2017) *Agile Testing Foundations*, BCS Learning & Development Ltd: Swindon, Storbritannien

Black, R. (2009) *Managing the Testing Process (3e)*, John Wiley & Sons: New York, NY

Buwalda, H. m. fl. (2001) *Integrated Test Design and Automation*, Addison Wesley: Reading, MA

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

Craig, R. och Jaskiel, S. (2002) *Systematic Software Testing*, Artech House: Norwood, MA

Crispin, L. och Gregory, J. (2008) *Agile Testing*, Pearson Education: Boston, MA

Fewster, M. och Graham, D. (1999) *Software Test Automation*, Addison Wesley: Harlow, Storbritannien

Gilb, T. och Graham, D. (1993) *Software Inspection*, Addison Wesley: Reading, MA

Graham, D. och Fewster, M. (2012) *Experiences of Test Automation*, Pearson Education: Boston, MA

Gregory, J. och Crispin, L. (2015) *More Agile Testing*, Pearson Education: Boston, MA

Jorgensen, P. (2014) *Software Testing, A Craftsman's Approach (4e)*, CRC Press: Boca Raton, FL

Kaner, C., Bach, J. och Pettichord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York, NY

Kaner, C., Padmanabhan, S. och Hoffman, D. (2013) *The Domain Testing Workbook*, Context-Driven Press: New York, NY

Kramer, A., Legeard, B. (2016) *Model-Based Testing Essentials: Guide to the ISTQB Certified Model-Based Tester: Foundation Level*, John Wiley & Sons: New York, NY

Myers, G. (2011) *The Art of Software Testing*, (3e), John Wiley & Sons: New York, NY

Sauer, C. (2000) "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research", *IEEE Transactions on Software Engineering*, *Volume 26, Issue 1, ss 1–*

Shull, F., Rus, I., Basili, V. juli 2000. "How Perspective-Based Reading can Improve Requirement Inspections." *IEEE Computer, Volume 33, Issue 7, ss 73–79*

van Veenendaal, E. (red.) (2004) *The Testing Practitioner* (kapitel 8–10), UTN Publishers: Nederländerna

Wiegers, K. (2002) *Peer Reviews in Software*, Pearson Education: Boston, MA

Weinberg, G. (2008) *Perfect Software and Other Illusions about Testing*, Dorset House: New York, NY

Övriga resurser (som inte refereras direkt i den här kursplanen)

Black, R., van Veenendaal, E. och Graham, D. (2012) *Foundations of Software Testing: ISTQB Certification (3e)*, Cengage Learning: London, Storbritannien

Hetzel, W. (1993) *Complete Guide to Software Testing (2e)*, QED Information Sciences: Wellesley, MA

Spillner, A., Linz, T., och Schaefer, H. (2014) *Software Testing Foundations (4e)*, Rocky Nook: San Rafael, CA

8 Bilaga A – Bakgrund till kursplanen

Dokumentets historik

Detta dokument är en svensk översättning av den internationella kursplanen (eng. Syllabus) inklusive anpassning till svensk terminologi.

Den internationella förlagan (Syllabus) förbereddes mellan 2014 och 2018 av en arbetsgrupp bestående av medlemmar som utsetts av International Software Testing Qualifications Board (ISTQB). 2018-versionen granskades inledningsvis av representanter från alla ISTQB-medlemsstyrelser, och därefter av representanter som utsetts bland internationella testare av programvara.

Mål för kvalifikationen på grundnivå

- Att skapa erkännande för testning som en viktig och professionell specialisering inom programvaruteknik
- Att utgöra ett standardramverk för utvecklingen av testarnas karriärer
- Att möjliggöra för professionellt kvalificerade testare att bli erkända hos arbetsgivare, kunder och kollegor, samt att stärka testarnas profil
- Att främja stabila och bra testmetoder inom alla delar av programvarudiscipliner
- Att identifiera testområden som är av betydelse och har värde för industrin
- Att möjliggöra för programvaruleverantörer att anlita certifierade testare och på så sätt få kommersiella fördelar över sina konkurrenter genom att informera om sin anställningspolicy vad gäller testare
- Att ge testare och andra med testintresse en möjlighet att få en internationellt erkänd kompetens inom området

Målsättningen med den internationella examen

- Att kunna jämföra kunskaper om testning i olika länder
- Att göra det enklare för testare att ta jobb över gränserna
- 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 samt att öka förståelsen om testning för alla
- Att lyfta fram testning som ett yrke i fler länder
- Att göra det möjligt för testare att få en erkänd examinering på sitt modersmål
- Att göra det möjligt att utbyta kunskaper och resurser mellan olika länder
- Att skapa ett internationellt erkännande av testare och denna examinering genom att deltagande sker från många länder

Förutsättningar för den här examen

Förutsättning för att genomföra ISTQB-examineringen för certifierade testare på grundnivån är att kandidaterna har intresse för testning av programvarutestning. Vi rekommenderar dock starkt att kandidaterna också:

- Har någon bakgrund inom antingen utveckling eller testning av programvara, till exempel sex månaders erfarenhet som system- eller användaracceptanstestare eller programvaruutvecklare
- Deltar i en kurs som har ackrediterats av ett ISTQB-erkänt medlemsorgan enligt ISTQB-standarderna. När det gäller Sverige är det SSTB.

Bakgrund och historik för certifikatet i programvarutestning

Den oberoende certifieringen av programvarutestare började i Storbritannien med British Computer Society's Information Systems Examination Board (ISEB), en styrelse för programvarutestning som inrättades 1998 (www.bcs.org.uk/iseb). 2002 började ASQF i Tyskland stötta ett tyskt kvalifikationsprogram för testare (www.asqf.de). Kursplanerna från ISEB och ASQF har varit bas vid framtagning av den internationella ISTQB-Syllabus. Denna har sedan omorganiserats, uppdaterats och nya områden har tillförts. Tonvikten är lagd på de områden som ger mest praktiskt stöd åt testarna.

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

I varje medlemsland styrs lokala aspekter av ett nationellt eller regionalt ISTQB-erkänt organ för programvarutestning. 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 examineringar.

9 Bilaga B – Utbildningsmål/kognitiva kunskapsnivåer

Följande utbildningsmål har definierats som gällande för den här kursplanen. Varje ämne i kursplanen kommer att examineras enligt de aktuella utbildningsmålen.

Nivå 1: Komma ihåg (K1)

Kandidaten ska kunna känna igen, minnas och komma ihåg ett uttryck eller ett begrepp.

Nyckelord: Komma ihåg, minnas, känna igen, kunna

Exempel:

Kan känna igen definitionen av ”felsymptom” som:

- ”Utebliven leverans av en tjänst till en slutanvändare eller annan intressent” eller
- ”Avvikelse från komponentens eller systemets förväntade leverans, beteende eller resultat”

Nivå 2: Förstå (K2)

Kandidaten kan välja orsaker eller förklaringar till påståenden som rör ämnet, och kan sammanfatta, jämföra, klassificera, kategorisera och ge exempel på testkonceptet.

Nyckelord: Sammanfatta, generalisera, abstrakt, klassificera, jämföra, kartlägga, kontrastera, exemplifiera, tolka, översätta, representera, uttyda, dra slutsatser, kategorisera, skapa modeller

Exempel:

Kan förklara orsaken till varför testanalys och testdesign bör genomföras så tidigt som möjligt:

- För att upptäcka defekter när de är billigare att åtgärda
- För att upptäcka de viktigaste defekterna först

Kan förklara likheter och skillnader mellan integrations- och systemtestning:

- Likheter: testobjekten för både integrationstestning och systemtestning inkluderar mer än en komponent, och både integrationstestning och systemtestning kan innehålla icke-funktionella testtyper
- Skillnader: integrationstestning är inriktad mot gränssnitt och interaktioner, och systemtestning är inriktat mot systemtäckande aspekter, till exempel end-to-end-hantering

Nivå 3: Tillämpa (K3)

Kandidaten kan välja rätt tillämpning av ett koncept eller en teknik och tillämpa den i ett givet sammanhang.

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

Exempel:

- Kan identifiera gränsvärden för giltiga och ogiltiga klasser
- Kan välja testfall från ett givet tillståndsövergångsdiagram för att täcka alla övergångar

Referens (för utbildningsmålens kognitiva nivåer)

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

10 Bilaga C – Release Notes

ISTQB:s syllabus på Foundation Level 2018, som översatts till svensk kursplan av SSTB, är en stor uppdatering och omarbetning av versionen från 2011. Därför finns det inte något detaljerad beskrivning per kapitel och avsnitt här, bara en sammanfattning av de största ändringarna. ISTQB tillhandahåller dessutom i ett separat dokument [1)] spårbarheten mellan utbildningsmålen (LO) i 2011-versionen av syllabus på Foundation Level och utbildningsmålen i 2018-versionen av Syllabus, så att man kan se vad som lagts till, uppdaterats eller tagits bort.

I början av 2017 hade mer än 550 000 personer i fler än 100 länder examinerat sig på Foundation Level, och antalet certifierade testare världen över är nu mer än 500 000. Förutsatt att alla har läst Syllabus, eller dess översättningar, på Foundation Level för att klara examineringen, gör det troligtvis denna Syllabus till det mest lästa dokumentet om testning av programvara någonsin!

Denna stora uppdatering har gjorts med respekt för detta arv och för att öka det värde ISTQB kan ge till kommande 500 000 personer i den globala testgemenskapen.

I den här versionen har alla utbildningsmål (LO) redigerats för att bli atomära. Detta för att skapa en tydlig spårbarhet från varje utbildningsmål och innehållsavsnitt (och examineringsfrågor) som är relaterade till det aktuella utbildningsmålet samt för att ha tydlig spårbarhet från innehållsavsnitten (och examineringsfrågorna) tillbaka till det tillhörande utbildningsmålet. Dessutom har tidsavsättningarna för kapitlen gjorts mer realistiska än i 2011-versionen av kursplanen genom att använda heuristik och formler från andra ISTQB-kursplaner, som bygger på en analys av utbildningsmålen för täckning för varje kapitel.

Även om detta är en kursplan på Foundation Level som innehåller bästa praxis och beprövade tekniker har vi gjort ändringar för att modernisera presentationen av materialet, särskilt avseende utvecklingsmetoder för programvara (till exempel scrum och continuous deployment) och teknologi (till exempel sakernas internet). Vi har uppdaterat referensstandarderna för att göra dem mer moderna enligt följande:

1. ISO/IEC/IEEE 29119 ersätter IEEE-standard 829.
2. ISO/IEC 25010 ersätter ISO 9126.
3. ISO/IEC 20246 ersätter IEEE 1028.

Eftersom ISTQB-portföljen har vuxit dramatiskt under det senaste årtiondet har vi lagt till omfattande korsreferenser till relaterat material i andra ISTQB-kursplaner i förekommande fall, och noga granskat materialet för anpassning till alla kursplaner och till ISTQB-ordlistan. Målet är att göra den här versionen enkel att läsa, förstå och översätta, med fokus på ökad praktisk användbarhet och balansen mellan kunskaper och färdigheter.

[1)] En detaljerad analys av förändringarna i den här versionen finns i ISTQB Certified Tester Foundation Level Overview 2018.

11 Index