

ISTQB® Certified Tester

**Advanced Level
Test Analyst
Kursplan**

Version 3.1.0

Swedish Software Testing Board

International Software Testing Qualifications Board



Copyrightmeddelande

Det här dokumentet får kopieras delvis eller i sin helhet förutsatt att källan uppges.

Copyrightmeddelande © International Software Testing Qualifications Board (hädanefter kallat ISTQB®)

ISTQB® är ett registrerat varumärke som tillhör International Software Testing Qualifications Board. SSTB är ett registrerat varumärke som tillhör Swedish Software Testing Board.

Advanced Level Test Analyst Sub Working Group: Judy McKay (Chair), Graham Bath

For the 2012 syllabus: Judy McKay, Mike Smith, Erik van Veenendaal

For the 2019 syllabus: Graham Bath (vice-chair), Rex Black, Judy McKay, Kenji Onoshi, Mike Smith (chair), Erik van Veenendaal

För den svenska versionen: Daniel Nordström, Dan Söderlund, Beata Karpinska, Ingvar Nordström, Simon Säter

Författarna och ISTQB® har kommit överens om att följande villkor ska gälla:

- Alla individer eller utbildningsföretag får använda den här kursplanen som underlag för en utbildning om författarna, ISTQB® och SSTB anges som källa och copyrightinnehavare för kursplanen och under förutsättning att eventuell reklam för en sådan utbildning endast får nämna kursplanen efter officiell ackreditering av utbildningsmaterialet av ett ISTQB-erkänt medlemsorgan.
- Alla individer eller grupper av individer får använda den här kursplanen som underlag för artiklar, böcker eller andra härledda verk om författarna, ISTQB® och SSTB anges som källa och copyrightinnehavare till kursplanen.

Revisionshistorik

Version	Date	Remarks
V3.1	2022-02-13	SSTB-release av Kursplan CTAL-TA 3.1

Innehållsförteckning

Revisionshistorik.....	3
Innehållsförteckning.....	4
Tillkännagivande.....	6
0. Inledning	7
0.1 Syftet med kursplanen	7
0.2 Certifierad testare på Advanced Level inom programvarutestning.....	7
0.3 Utbildningsmål som kan examineras och kognitiva kunskapsnivåer	7
0.4 Examen för Advanced Level Test Analyst	8
0.5 Förutsättningar för examen	8
0.6 Speciell erfarenhet	8
0.7 Ackreditering	8
0.8 Detaljnivå.....	8
0.9 Kursplanens upplägg	9
1. Testanalytikerns arbetsuppgifter i testprocessen - 150 minuter.....	10
1.1 Introduktion	11
1.2 Testning i programvarans utvecklingslivscykel	11
1.3 Testanalys.....	12
1.4 Testdesign.....	13
1.4.1 Lågnivåtestfall och högnivåtestfall	14
1.4.2 Design av testfall	15
1.5 Testimplementering.....	16
1.6 Testexekvering	18
2. Testanalytikerns arbetsuppgifter i riskbaserad testning – 60 minuter,	19
2.1 Introduktion	20
2.2 Riskidentifiering	20
2.3 Riskbedömning	21
2.4 Riskreducering	21
2.4.1 Prioritering av testerna.....	22
2.4.2 Anpassa testningen för framtida testcykler.....	22
3. Testtekniker - 630 minuter.	23
3.1 Introduktion	24
3.2 Black-Box-testtekniker	24
3.2.1 Ekvivalensklassindelning	24
3.2.2 Gränsvärdesanalys.....	26
3.2.3 Testning med hjälp av beslutstabeller	27
3.2.4 Tillståndsbaserad testning	28
3.2.5 Klassifikationsträdsteknik.....	30
3.2.6 Parvis testning	31
3.2.7 Användningsfallsbaserad testning	32
3.2.8 Kombination av tekniker	33
3.3 Erfarenhetsbaserade testtekniker	34
3.3.1 Felgissning.....	34
3.3.2 Checklistebaserad testning.....	35
3.3.3 Utforskande testning.....	36
3.3.4 Defektbaserade testtekniker	37
3.4 Tillämpning av den mest lämpliga tekniken	38
4. Testning av programvarans kvalitetsegenskaper - 180 minuter.....	39
4.1 Introduktion	40
4.2 Kvalitetsegenskaper för testning av affärsdomäner.....	41
4.2.1 Testning av funktionell korrekthet.....	41
4.2.2 Testning av funktionell ändamålsenlighet.....	41

4.2.3	Testning av funktionell kompletthet	41
4.2.4	Interoperabilitetstestning.....	42
4.2.5	Användbarhetsutvärdering.....	43
4.2.6	Portabilitetstestning	45
5.	Granskningar - 120 minuter.	47
5.1	Introduktion	48
5.2	Använda checklistor i granskning	48
5.2.1	Kravgranskningar.....	48
5.2.2	Granskningar av användarberättelser	49
5.2.3	Att skraddarsy checklistor.....	49
6.	Testverktyg och automatisering - 90 minuter.	51
6.1	Introduktion	52
6.2	Nyckelordsdriven Testning.....	52
6.3	Typ av testverktyg	53
6.3.1	Testdesignverktyg.....	53
6.3.2	Verktyg för förberedelse av testdata.....	53
6.3.3	Automatiserade testexekveringsverktyg.....	53
7.	Referenser	55
7.1	Standarder.....	55
7.2	ISTQB® och IREB Dokument	55
7.3	Böcker och Artiklar	55
7.4	Andra referenser	57
8.	Appendix A.....	58
9.	Index	59

Tillkännagivande

Det engelska originalet till detta dokument har tagits fram av Test Analyst task force of the International Software Testing Qualifications Board Advanced Level Working Group: Mette Bruhn-Pedersen (Working Group Chair); Matthias Hamburg (Product Owner); Wim Decoutere, István Forgács, Adam Roman, Jan Sabak, Marc-Florian Wendland (Authors).

0. Inledning

0.1 Syftet med kursplanen

Den här kursplanen utgör grunden för certifiering på ISTQB® Advanced Level, Test Analyst. 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 systemutvecklare 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 Advanced Level inom programvarutestning

Advanced Level Core består av tre separata kursplaner med följande roller:

- Test Manager
- Test Analyst
- Technical Test Analyst

Det separata dokumentet ISTQB® Advanced Level Overview 2019 [ISTQB_AL_OVIEW] innehåller följande information:

- Överordnat utbildningsresultat för varje Kursplan/Syllabus
- En matris som visar spårbarheten mellan verksamhetsresultat och utbildningsmål (LO)
- Sammanfattning för varje Kursplan/Syllabus
- Relationen mellan Kursplanerna/Syllabi

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 för Test Analyst.

Kunskapsnivåerna för de specifika utbildningsmålen visas i början av varje kapitel och klassificeras enligt följande:

- K2: Förstå
- K3: Tillämpa
- K4: Analysera

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

0.4 Examen för Advanced Level Test Analyst

Examen för Advanced Level, Test Analyst 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 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 poängen uppnås.

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 Förutsättningar för examen

Certifikat på Foundation Level (CTFL) är en förutsättning för att genomföra examen på Advanced Level Test Analyst.

0.6 Speciell erfarenhet

Inget av utbildningsmålen i Advanced Test Analyst förutsätter speciell erfarenhet.

0.7 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 följas upp med en ISTQB®-examinering i anslutning till kursen.

0.8 Detaljnivå

Detaljnivån i den här kursplanen möjliggör skapandet av kurser och examineringar som stämmer överens internationellt. För att uppnå det målet innehåller kursplanen:

- Allmänna instruktionsmål som beskriver syftet med Advanced Level Test Analyst
- En lista med termer som eleverna måste komma ihåg
- Utbildningsmål (Learning Objectives) för varje kunskapsområde som beskriver det kognitiva utbildningsresultat som ska uppnås
- En beskrivning av viktiga koncept, inklusive referenser till källor som godkänd litteratur och standarder

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

0.9 Kursplanens upplägg

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 20,5 timmars undervisning som fördelas över de sex kapitlen enligt följande:

- Kapitel 1: Testanalytikerns arbetsuppgifter i testprocessen (150 minuter)
- Kapitel 2: Testanalytikerns arbetsuppgifter i riskbaserad testning (60 minuter)
- Kapitel 3: Testtekniker (630 minuter)
- Kapitel 4: Testning av kvalitetsegenskaper för programvara (180 minuter)
- Kapitel 5: Granskningar (120 minuter)
- Kapitel 6: Testverktyg och automatisering (90 minuter)

1. Testanalytikerns arbetsuppgifter i testprocessen - 150 minuter.

Nyckelord

avslutskriterier, högnivåtestfall, lågnivåtestfall, test, testanalys, testbas, testdata, testdesign, testexekvering, testexekveringsschema, testimplementation, testprocedur, testsvit, testvillkor

Utbildningsmål för testanalytikerns arbetsuppgifter i testprocessen

1.1 Introduktion

Inga utbildningsmål

1.2 Testning i programvarans utvecklingslivscykel

TA-1.2.1 (K2) Förklara hur och varför tidpunkten och engagemangsnivån för testanalytikern varierar när arbete med olika utvecklingslivscykelmodeller för programvaran sker

1.3 Testanalys

TA-1.3.1 (K2) Sammanfatta de lämpliga arbetsuppgifterna för testanalytikern när analysaktiviteter utförs

1.4 Testdesign

TA-1.4.1 (K2) Förklara varför testvillkor borde vara förstådda av intressenter

TA-1.4.2 (K4) För ett givet projektscenario selektera den passande designnivån för testfall (hög- eller lågnivå)

TA-1.4.3 (K2) Förklara problemen att ta hänsyn till i testfallsdesign

1.5 Testimplementering

TA-1.5.1 (K2) Sammanfatta de lämpliga arbetsuppgifterna för testanalytikern när testimplementationsaktiviteter utförs

1.6 Testexekvering

TA-1.6.1 (K2) Sammanfatta de lämpliga arbetsuppgifterna för testanalytikern när testexekveringsaktiviteter utförs

1.1 Introduktion

I ISTQB® Foundation Level-kursplanen beskrivs testprocessen med följande aktiviteter:

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

I den här kursplanen beskrivs de aktiviteter som har särskild betydelse för testanalytiker mer ingående. Detta ger ytterligare förbättringar av testprocessen för att bättre anpassa olika modeller för programvarors utvecklingslivscykel (SDLC).

Att bestämma de lämpliga testerna, designa, implementera och sedan exekvera dem är det främsta fokusområdena för en testanalytiker. Även om det är viktigt att förstå de andra stegen i testprocessen är majoriteten av testanalytikerns arbete vanligtvis fokuserad på följande aktiviteter:

- Testanalys
- Testdesign
- Testimplementation
- Testexekvering

De andra aktiviteterna i testprocessen är tillräckligt beskrivet på Foundation Level och behöver inte ytterligare förklaring på denna nivå.

1.2 Testning i programvarans utvecklingslivscykel

Den övergripande SDLC bör övervägas när en teststrategi definieras. Tidpunkten för testanalytikerns deltagande varierar för olika SDLC. Omfattningen av involvering, nödvändig tid, tillgänglig information och förväntningar kan också variera. Testanalytikern måste vara medveten om vilken typ av information som ska tillhandahållas till andra relaterade organisatoriska roller såsom:

- Kravhantering – återkoppling på resultat från kravgranskningar
- Projektledning - input från tidsplan
- Konfiguration och ändringshantering – testresultat från verifiering av bygge, versionshanteringsinformation
- Programvaruutveckling - anteckningar av funna fel
- Programvaruunderhåll - rapporter om fel, felrättnings effektivitet och omtestning,
- Teknisk support - rätt dokumentation för åtgärder och kända problem
- Produktion av teknisk dokumentation (till exempel databasdesignspecifikationer, testmiljödokumentation) - input till dessa dokument samt teknisk granskning av dokumenten

Testaktiviteter måste anpassas till den valda SDLC vars karaktär kan vara sekventiell, iterativ, inkrementell eller en hybridblandning av dessa. Till exempel kan i den sekventiella V-modellen testprocessen på systemtestnivån anpassas enligt följande:

- Systemtestplanering sker samtidigt med projektplanering, och testövervakning och kontroll fortsätter till testavslut. Detta kommer att påverka planerad information som tillhandahålls av testanalytikern för projektledningsändamål.
- Systemtestanalys och design anpassas till dokument som systemkravspecifikation, system- och arkitektonisk (hög nivå) designspecifikation och komponent (låg nivå) designspecifikation.
- Implementering av systemtestmiljön kan påbörjas under systemdesign, även om huvuddelen av det vanligtvis skulle inträffa samtidigt med kodning och komponenttestning, med arbete på

systemtestimplementeringsaktiviteter som ofta sträcker sig tills bara några dagar före genomförandet av systemtestexekvering.

- Systemtestexekvering påbörjas när startkriterier är uppfyllda (eller inte om nödvändigt,) vilket vanligtvis innebär att åtminstone komponenttestning och ofta även komponentintegrationstestning har uppfyllt sina avslutskriterier. Systemtestexekveringen fortsätter ända tills systemtestningens avslutskriterier är uppfyllda.
- Systemtestavslutningsaktiviteter sker efter att dess avslutskriterier är uppfyllda.

Iterativa och inkrementella modeller kanske inte följer samma arbetsordning och kan utesluta vissa aktiviteter. Till exempel kan en iterativ modell använda en reducerad uppsättning av testaktiviteter för varje iteration. Testanalys, design, implementering och exekvering kan genomföras för varje iteration, medan högnivåplanering görs i början av projektet och avslutningsuppgifter görs i slutet.

I ett agilt projekt är det vanligt att använda en mindre formaliserad process och en mycket närmare arbetsrelation med projektintressenter som lättare tillåter förändringar att ske inom projektet. Det kan saknas en väldefinierad roll för testanalytikern. Det finns mindre omfattande testdokumentation och kommunikationen är kortare och mer frekvent.

Agil programvaruutveckling inkluderar testning från början. Detta börjar vid inledningen av produktutvecklingen när utvecklarna utför sitt ursprungliga arkitektur- och designarbete. Granskning får inte formaliseras men är kontinuerlig under tiden som programvaran utvecklas. Involvering förväntas vara genom hela projektet och testanalytikerns uppgifter förväntas göras av teamet.

Iterativa och inkrementella modeller omfattar allt från agil programvaruutveckling, där det finns en förväntan på ändringar när kundkraven utvecklas, till hybridmodeller som t.ex. iterativ/inkrementell utveckling kombinerat med en V-modell. I sådana hybridmodeller, ska testanalytikern förvänta sig att vara delaktig i planerings- och designaspekterna, men sedan byta till en mer interaktiv roll under tiden som programvaran designas, utvecklas och testas.

Oavsett vilken SDLC som används, behöver testanalytikern förstå förväntningarna på deltagandet samt tidpunkten för detta deltagande. Testanalytikern bidrar effektivt till programvarukvaliteten genom att anpassa sina aktiviteter och sin tidpunkt för deltagandet med en specifik SDLC snarare än att hålla sig fast vid en fördefinierad rollmodell.

1.3 Testanalys

Under testplaneringen definieras testprojektets omfattning. Under testanalysen bidrar testanalytikern med följande:

- Analysera testbasen
- Identifiera defekter av olika typer i testbasen
- Identifiera och prioritera testvillkor och features som ska testas
- Identifiera dubbelriktad spårbarhet mellan varje del av testbasen och de tillhörande testvillkoren
- Utföra uppgifter som är associerade med riskbaserad testning (se Kapitel 2)

För att testanalytikern effektivt ska kunna fortsätta med testanalysen bör följande startkriterier uppfyllas:

- Det finns en kunskapsbank (till exempel, krav, användarberättelse) som beskriver testobjektet som kan fungera som dess testbas (se [ISTQB_FL_SYL] Kapitel 1.4.2 och 2.2 eller en lista över andra möjliga testbaser).
- Denna testbas har granskats och godkänts med godtagbara resultat och har uppdaterats efter behov efter granskningen. Observera att om högnivåtestfall ska definieras (se Kapitel 1.4.1) kan testbasen inte behöva definieras fullt ut än. I agil programvaruutveckling kommer denna

granskningscykel att vara iterativ eftersom användarberättelserna förfinas i början av varje iteration.

- Det finns en godkänd budget och tidsplan tillgänglig för att utföra de återstående testuppgifterna för detta testobjekt.

Testvillkor är vanligtvis identifierade genom analys av testbasen i samband med testmålen (som är definierade i testplaneringen). I vissa situationer där dokumentationen kan vara gammal eller obefintlig, kan testvillkoren identifieras genom diskussion med relevanta intressenter (till exempel i workshops eller under iterationsplanering). I agil programvaruutveckling används ofta acceptanskriterier, som definieras som en del av användarberättelserna, som grund för testdesignen.

Även om testvillkoren vanligtvis är specifika för det objekt som testas, finns det några standardöverväganden för testanalytikern.

- Det rekommenderas vanligtvis att definiera testvillkoren på olika detaljnivåer. Ursprungligen identifieras högnivåvillkor för att definiera allmänna mål för testning, till exempel "funktionaliteten av skärm x". Därefter identifieras mer detaljerade villkor som grunden för specifika testfall, till exempel "skärm x avvisar ett kontonummer som saknar en siffra jämfört med det korrekta antalet". Att använda denna typ av hierarkisk metod för att definiera testvillkoren kan hjälpa till att säkerställa att täckningsgraden är tillräcklig för högnivåobjekt. Detta tillvägagångssätt gör det också möjligt för en testanalytiker att börja arbete med att definiera högnivåtestvillkor för användarberättelser som ännu inte har förfinats.
- Ifall produktrisker har definierats bör testvillkoren som är nödvändiga för att hantera varje produktrisk identifieras och spåras tillbaka till detta riskobjekt.

Tillämpningen av testtekniker (som identifieras i teststrategin och/eller testplanen) kan vara till hjälp i testanalysaktiviteter och kan användas för att underlätta följande mål:

- Identifiera testvillkor
- Minska sannolikheten för utelämnande av viktiga testvillkor
- Definiera mer exakta och noggranna testvillkor
- Efter det att testvillkoren har identifierats och förfinats kan granskning av dessa villkor göras med intressenterna för att säkerställa att kraven är tydligt förstådda och att testningen är i linje med projektets mål.

Vid slutet av testanalysaktiviteterna för ett visst område (till exempel en specifik funktion) vet testanalytikern vilka specifika tester som måste designas för det området.

1.4 Testdesign

Testprocessen, som fortfarande följer den omfattning bestämd under testplaneringen, fortsätter med att testanalytikern designar de tester som ska implementeras och exekveras. Testdesign inkluderar följande aktiviteter:

- Bestämma i vilka testområden lågnivå- eller högnivåtestfall är lämpliga
- Bestämma testteknik(er) som gör det möjligt att uppnå den nödvändiga täckningsgraden. De tekniker som kan användas bestäms under testplaneringen.
- Använda testtekniker för att designa testfall och uppsättningar av testfall som täcker de identifierade testvillkoren
- Identifiera nödvändig testdata för att stöda testvillkor och testfall
- Designa testmiljön och identifiera nödvändig infrastruktur inklusive verktyg
- Identifiera dubbelriktad spårbarhet (till exempel mellan testbas, testvillkor och testfall)

Prioriteringskriterier som identifierats under riskanalys och testplanering bör tillämpas under hela processen, från analys och design till implementering och exekvering.

Beroende på vilka typer av tester som designas kan ett av startkriterierna för testdesign vara tillgängligheten av verktyg som kommer att användas under designarbetet.

Under testdesign måste testanalytikern åtminstone tänka på följande:

- Vissa testelement behandlas bättre genom att bara definiera testvillkoren snarare än att gå vidare i definitionen av testskript, som ger sekvensen av instruktioner som krävs för att exekvera ett test. I det här fallet bör testvillkoren definieras för att användas som en guide för oskriptad testning
- Kriterierna för godkänt/underkänt ska lätt kunna identifieras.
- Tester ska designas så att de kan förstås av andra testare, inte bara författaren. Om författaren inte är den person som exekverar testet, måste andra testare läsa och förstå tidigare specificerade tester för att förstå testmålen och testets relativa betydelse.
- Tester måste också vara förståeliga för andra intressenter, till exempel utvecklare, (som kan granska testerna), och andra (som kan behöva godkänna testerna).
- Tester ska täcka alla typer av interaktion med testobjektet och bör inte begränsas till interaktion med människor genom det användarsynliga gränssnittet. De kan till exempel också inkludera interaktion med andra system och tekniska eller fysiska händelser. (se [IREB_CPRE] för mer information).
- Test ska designas för att testa gränssnitten mellan olika testobjekt, såväl som själva objektens beteende.
- Testdesigninsatser måste prioriteras och balanseras för att anpassa sig till risknivån och affärsvärdet.

1.4.1 Lågnivåtestfall och högnivåtestfall

Ett av testanalytikers jobb är att bestämma den bästa designnivån för testfall för en given situation. Lågnivå- och högnivåtestfall beskrivs i [ISTQB_FL_SYL]. Några av fördelarna och nackdelarna med att använda dessa beskrivs i följande listor:

Lågnivåtestfall ger följande fördelar:

- Oerfaren testpersonal kan förlita sig på detaljerad information som tillhandahålls inom projektet. Lågnivåtestfall ger all specifik information och procedurer som krävs för att testaren kan exekvera testfallet (inklusive eventuella datakrav) och för att verifiera de faktiska resultaten.
- Tester kan exekveras om av olika testare och bör då uppnå samma testresultat.
- Ej uppenbara fel i testbasen kan upptäckas.
- Detaljeringsnivån möjliggör en oberoende verifiering av testerna, till exempel granskning, om det behövs.
- Tiden som används för implementering av automatiserade testfall kan minskas.

Lågnivåtestfall har följande nackdelar:

- De kan kräva en stor insats, både för skapande och underhåll.
- De har en tendens att begränsa testarens påfattighet under exekvering.
- De kräver att testbasen är väldefinierad.
- Deras spårbarhet till testvillkoren kan kräva en större insats än med högnivåtestfall.

Högnivåtestfall ger följande fördelar:

- De ger riktlinjer för vad som ska testas och tillåter testanalytikern att variera faktiska data eller till och med proceduren som ska följas vid testexekvering
- De kan ge bättre risktäkningsgrad än lågnivåtestfall eftersom de kommer att variera något varje gång de exekveras.
- De kan definieras tidigt i kravprocessen.

- De använder sig av testanalytikerens erfarenhet av både testning och testobjektet när testet exekveras.
- De kan definieras när ingen detaljerad och formell dokumentation krävs.
- De är bättre lämpade för återanvändning i olika testcykler när olika testdata kan användas.

Högnivåtestfall har följande nackdelar:

- De är mindre reproducerbara, vilket gör verifiering svår. Detta beror på att de saknar den detaljerade beskrivningen som finns i lågnivåtestfall.
- Mer erfaren testpersonal kan behövas för att exekvera dem
- Vid automatisering baserat på högnivåtestfall kan bristen på detaljer leda till validering av fel resultat, eller saknade element som skulle ha validerats.

Högnivåtestfall kan användas för att utveckla lågnivåtestfall när kraven blir mer definierade och stabila. I det här fallet skapas testfallet sekventiellt och strömmar från högnivå till lågnivå med endast lågnivåtestfall som används för exekvering.

1.4.2 Design av testfall

Testfall designas genom stegvis utarbetande och förfining av de identifierade testvillkoren med hjälp av testtekniker (se Kapitel 3). Testfall ska vara repeterbara, verifierbara och spårbara tillbaka till testbasen (till exempel kraven).

Testdesign inkluderar identifiering av följande:

- Mål (dvs. det observerbara, mätbara målet för testexekvering)
- Förutsättningar, till exempel antingen projekt- eller lokala testmiljökrav och planerna för deras leverans, systemets status före testexekvering etc.
- Testdatakrav (både indata för testfallet såväl som data som måste existera i systemet för att testfallet ska kunna exekveras)
- Förväntade resultat med uttryckliga kriterier för godkännande/underkännande
- Sluttillstånd, som påverkat data, systemets tillstånd efter testexekvering, triggers för påföljande bearbetning etc.

En speciell utmaning kan vara definitionen av det förväntade resultatet av ett test. Att beräkna detta manuellt är ofta tråkigt och felbenäget. Om möjligt är det att föredra att hitta eller skapa ett automatiserat testorakel. Vid framtagning av det förväntade resultatet är testerna intresserade, inte bara av information på skärmen, utan också av data och miljösluttillstånd. Om testbasen är tydligt definierad, bör identifiering av rätt resultat, teoretiskt, vara enkelt. Testbasdokumentationen kan dock vara vag, motsägelsefull, sakna täckning av viktiga områden eller helt saknas. I sådana fall måste en testanalytiker vara expert inom ämnesområdet eller ha tillgång till den. Även när testbasen är väl specificerad kan komplexa interaktioner mellan komplexa stimuli och svar göra definitionen av de förväntade resultaten svår; därför är ett testorakel viktigt. I agil programvaruutveckling kan testoraklet vara produktägaren. Testfallsexekvering utan något sätt att bestämma korrektheten av de faktiska resultaten har ett väldigt litet mervärde, vilket ofta ger upphov till felaktiga testrapporter eller dåligt förtroende för systemet.

Aktiviteterna som beskrivs ovan kan tillämpas på alla testnivåer, även om testbasen kommer att variera. När man analyserar och designar tester är det viktigt att komma ihåg målet såväl som syftet med testet. Detta hjälper till att bestämma detaljnivån som krävs, såväl som alla nödvändiga verktyg (till exempel drivrutiner och stubbar på komponenttestnivån).

Under utvecklingen av testvillkor och testfall skapas vanligtvis en viss mängd dokumentation, vilket resulterar i testarbetsprodukter. I praktiken varierar omfattningen av dokumentation av testarbetsprodukter mycket. Detta kan påverkas av något av följande:

- Projektrisker (vad som måste/inte måste dokumenteras)

- Det extra värdet som dokumentationen ger projektet
- Standarder som ska följas och/eller regleringar som ska uppfyllas
- SDLC eller tillvägagångssätt som används (till exempel ett agilt angreppssätt som eftersträvar ”tillräckligt” med dokumentation)
- Kravet på spårbarhet från testbasen genom testanalys och design

Beroende på testningens omfattning så adresserar testanalys och design också kvalitetsegenskaperna för testobjektet (en). ISO 25010-standardens [ISO25010] är en användbar referens. Vid testning av hårdvaru/programvarusystem kan ytterligare egenskaper gälla.

Aktiviteterna för testanalys och testdesign kan förbättras genom att sammanfoga dem med granskningar och statisk analys. I själva verket är genomförandet av testanalysen och testdesignen ofta en form av statisk testning eftersom problem kan upptäckas i testbasdokumenten under denna aktivitet.

Testanalys och testdesign baserat på kravspecifikationen är ett utmärkt sätt att förbereda för ett kravgransningsmöte. Att läsa kraven för att skapa tester kräver att man förstår dem och kan bestämma ett sätt att bedöma uppfyllandet av kravet. Denna aktivitet upptäcker ofta saknade krav, krav som är otydliga, inte kan testas eller har odefinierade acceptanskriterier. På liknande sätt kan testarbetsprodukter som testfall, riskanalyser och testplaner underkastas granskningar.

Under testdesign kan de nödvändiga detaljerade testinfrastrukturkraven definieras, även om dessa i praktiken kanske inte slutförs förrän i testimplementeringen. Det måste hållas i åtanke att testinfrastrukturen innehåller mer än testobjekt och testvara. Till exempel kan infrastrukturkraven inkludera rum, utrustning, personal, programvara, verktyg, kringutrustning, kommunikationsutrustning, användarauktorisering och alla andra element som krävs för att genomföra testerna.

Avslutskriterierna för testanalys och testdesign kommer att variera beroende på projektparametrarna, men alla element som nämnts i dessa två sektioner bör beaktas för att inkluderas i de definierade avslutskriterierna. Det är viktigt att avslutskriterierna är mätbara och att all information som krävs för de efterföljande stegen har tillhandahållits och alla nödvändiga förberedelser har gjorts.

1.5 Testimplementering

Testimplementering förbereder testvaran som behövs för testexekvering baserat på testanalys och design. Det inkluderar följande aktiviteter:

- Utveckla testprocedurer, och kanske också skapa automatiserade testskript
- Organisera testprocedurer och automatiserade testskript (om det finns några) i testsviter för att exekveras i en specifik testgenomgång
- Konsultera testledaren för att prioritera testfall och testsviter som ska exekveras
- Skapa ett testexekveringsschema, inklusive resursallokering, för att möjliggöra testexekvering (se [ISTQB_FL_SYL] Kapitel 5.2.4)
- Slutföra förberedelse av testdata och testmiljöer
- Uppdatera spårbarheten mellan testbasen och testvaran som testvillkor, testfall, testprocedurer, testskript och testsviter.

Under testimplementeringen identifierar testanalytikern en effektiv exekveringsordning för testfall och skapar testprocedurer. Att definiera testprocedurer kräver noggrann identifiering av begränsningar och beroenden som kan påverka testexekveringsordningen. Testprocedurer dokumenterar initiala förutsättningar (till exempel laddning av testdata från ett databas) och alla aktiviteter som följer efter exekveringen (till exempel återställning av systemstatusen).

Testanalytikern identifierar testprocedurer och automatiserade testskript som kan grupperas (t.ex. relaterar alla till testningen av en särskilt affärsprocess på hög nivå), och organiserar dem i testsviter.

Testanalytikern sammanställer testsviter i ett testexekveringsschema för en effektiv testexekvering. Om en riskbaserad teststrategi används kan risknivån diktera exekveringsordningen för testfallen. Det kan också finnas andra faktorer som bestämmer exekveringsordningen, till exempel tillgängligheten till rätt personer, utrustning, data och funktionaliteten som ska testas.

Det är inte ovanligt att kod släpps i delar och testinsatsen måste koordineras med den sekvens som programvaran blir tillgänglig för testning. Särskilt i iterativa och inkrementella utvecklingsmodeller är det viktigt för testanalytikern att samordna med utvecklingsgruppen för att säkerställa att programvaran kommer att släppas till testning i en testbar ordning.

Detaljnivån och tillhörande komplexitet för arbete som utförs under testimplementering kan påverkas av detaljerna i testvillkoren och testfallen. I vissa fall gäller förordningar och testarbetsprodukterna ska vara bevis på typgodkännande enligt gällande standarder som USA:s standard DO-178C (i Europa ED 12C). [RTCA DO-178C/ED-12C].

Som nämnt ovan krävs testdata för de flesta tester, och i vissa fall kan dessa uppsättningar av data vara ganska stora. Under implementeringen skapar testanalytiker in- och miljödata för att ladda in i databaser och liknande. Dessa data måste vara "fit for purpose" för att möjliggöra upptäckter av fel. Testanalytiker kan också skapa data som kan användas med datadriven och nyckelordsdriven testning (se Kapitel 6.2) samt för manuell testning.

Testimplementering avser också testmiljön/testmiljöerna. Under denna aktivitet ska miljö/miljöerna konfigureras och verifieras innan testexekvering genomförs. En testmiljö som är "fit for purpose" är väsentlig, det vill säga testmiljön ska kunna möjliggöra exponering av de fel som finns närvarande under kontrollerad testning, fungera normalt när fel inte inträffar och på lämpligt sätt, om så krävs, replikera produktionen eller slutanvändarmiljön för högre testnivåer. Testmiljöändringar kan vara nödvändiga under testexekvering beroende på oväntade förändringar, testresultat eller andra omständigheter. Om miljöförändringar inträffar under exekvering är det viktigt att bedöma inverkan av ändringarna i tester som redan har exekverats.

Under testimplementering ska testanalytikern kontrollera att de ansvariga för skapandet och underhållet av testmiljön är kända och tillgängliga, och att all testvara och testverktyg och tillhörande processer är klara för användning. Detta inkluderar konfigurationshantering, felhantering och testloggning och -hantering. Testanalytikern måste dessutom verifiera procedurerna som samlar in data för att utvärdera aktuell status mot avslutskriterier och testresultatsrapportering.

Det är klokt att använda ett balanserat tillvägagångssätt för testimplementering som bestämts under testplanering. Till exempel blandas riskbaserade analytiska teststrategier ofta med reaktiva teststrategier. I detta fall allokeras en del av insatsen för testimplementering till tester som inte följer förutbestämda skript (oskriptad).

Oskriptad testning ska inte vara slumpmässig eller utan mål eftersom det kan vara oförutsägbart i varaktighet och täckning och ge ett lågt resultat i funna defekter. Snarare ska det genomföras i tidsbegränsade sessioner där var och en ges initial inriktning av en testcharter, men med friheten att avvika från charterns föreskrifter ifall potentiellt mer produktiva testmöjligheter upptäcks under loppet av sessionen. Under åren har testare utvecklat en mängd olika erfarenhetsbaserade testtekniker, såsom attacker [Whittaker03], felgissning [Myers11] och utforskande testning [Whittaker09]. Testanalys, testdesign och testimplementering sker fortfarande, men de sker främst under testexekvering.

När man följer sådana reaktiva teststrategier påverkar resultaten av varje test analysen, designen och implementeringen av de efterföljande testerna. Medan dessa strategier är lättviktiga och ofta effektiva för att hitta defekter, finns det några nackdelar inklusive följande:

- Expertis från testanalytikern är nödvändig
- Varaktighet kan vara svårt att förutspå
- Täckningsgrad kan vara svår att spåra
- Repeterbarhet kan gå förlorad utan bra dokumentation eller verktygsstöd

1.6 Testexekvering

Testexekvering utförs enligt testexekveringsschemat och inkluderar följande uppgifter: (se [ISTQB_FL_SYL])

- Exekvera manuella tester, inklusive utforskande testning
- Exekvera automatiserade tester
- Jämföra faktiska resultat med förväntade resultat
- Analysera avvikelser för att bestämma deras troliga orsaker
- Rapportera fel baserat på observerade felsymptom
- Logga de faktiska resultaten från testexekveringen
- Uppdatera spårbarheten mellan testbasen och testvaran för att bestämma testresultatet
- Exekvera regressionstestning

Testexekveringsuppgifterna som listas ovan kan utföras av antingen testaren eller testanalytikern.

Följande är typiska extra uppgifter som testanalytikern kan utföra:

- Känna igen defektkluster som kan indikera behovet av mer testning av en viss del av testobjektet
- Lämna förslag för framtida utforskande testningssessioner baserat på upptäckten från den utforskande testningen
- Identifiera nya risker från information som erhållits vid utförande av testexekveringsuppgifter
- Lämna förslag för att förbättra någon av arbetsprodukterna från testimplementeringsaktiviteten (till exempel förbättringar av testrutiner)

2. Testanalytikerns arbetsuppgifter i riskbaserad testning – 60 minuter,

Nyckelord

produktrisk, riskbaserad testning, riskidentifiering, riskreducering

Utbildningsmål för testanalytikerns arbetsuppgifter i riskbaserad testning

Testanalytikerns arbetsuppgifter i riskbaserad testning

TA-2.1.1 (K3) För en given situation, delta i riskidentifiering, utföra riskbedömning och föreslå lämplig riskreducering

2.1 Introduktion

Testansvariga har ofta det övergripande ansvaret för att upprätta och hantera en riskbaserad teststrategi. De kommer vanligtvis att begära att en testanalytiker deltar för att säkerställa att den riskbaserade strategin implementeras korrekt.

Testanalytikern ska vara aktivt inblandad i följande steg för riskbaserad testning:

- Riskidentifiering
- Riskbedömning
- Riskreducering

Dessa uppgifter exekveras iterativt i hela SDLC för att hantera nya risker, ändrade prioriteringar och för att regelbundet utvärdera och kommunicera riskstatus (se [vanVeenendaal12] och [Black02] för mer information). I agil programvaruutveckling kombineras ofta de tre uppgifterna i en så kallad risksession med fokus på antingen en iteration eller en release.

Testanalytiker bör arbeta inom den riskbaserade teststrategin som bestämts av den testansvariga för projektet. Testanalytikern ska bidra med sina kunskaper om affärsdomänrisker som finns i projektet, bland annat risker relaterade till funktionssäkerhet, affärs- och ekonomiska problem, och politiska faktorer.

2.2 Riskidentifiering

Genom att kräva deltagande av det största möjliga urvalet av intressenter kommer riskidentifikationsprocessen troligen att upptäcka det största möjliga antalet väsentliga risker.

Testanalytiker har ofta unik kunskap om det specifika affärsområde för det testade systemet. Detta innebär särskilt lämpligheten för följande uppgifter:

- Genomföra expertintervjuer med domänexperter och användare
- Genomföra oberoende utvärderingar
- Använda riskmallar
- Delta i riskworkshops
- Delta i brainstorming-sessioner med potentiella och nuvarande användare
- Definiera testchecklistor
- Använda tidigare erfarenhet av liknande system eller projekt

Testanalytiker bör i synnerhet arbeta nära med användare och andra domänexperter (till exempel kravingenjörer och affärsanalytiker) för att bestämma de områden med affärsrisk som bör tas upp under testning. I agil programvaruutveckling möjliggör denna nära relation till intressenter att regelbundna riskidentifieringar kan genomföras, exempelvis under iterationsplaneringsmöten.

Exempel på risker som kan identifieras i ett projekt inkluderar:

- Problem med funktionell korrekthet, till exempel felaktiga beräkningar
- Användbarhetsproblem, till exempel otillräckliga kortkommandon
- Portabilitetsproblem, till exempel oförmåga att installera en applikation på vissa plattformar

2.3 Riskbedömning

Medan riskidentifiering handlar om att identifiera så många relevanta risker som möjligt, är riskbedömning studien av dessa identifierade risker. Specifikt, kategorisera varje risk och bedöma risknivåerna.

Att bedöma risknivån innebär vanligtvis att, för varje riskobjekt, uppskatta sannolikheten för inträffande och konsekvens. Sannolikheten tolkas vanligtvis som hur troligt det är att det potentiella problemet kan existera i det system som testas och kommer att observeras när systemet är i produktion. Tekniska testanalytikern ska bidra till att hitta och förstå den potentiella sannolikheten för varje riskobjekt medan testanalytikern bidrar till att förstå den potentiella affärspåverkan av problemet om det skulle inträffa (i agil programvaruutveckling kan denna rollbaserade distinktion vara mindre stark).

Riskens påverkan tolkas ofta som allvarligheten av effekten på användarna, kunderna eller andra intressenter. Med andra ord, det härrör från affärsrisken. Testanalytikern ska bidra till att identifiera och utvärdera den potentiella affärsdomänen eller användarpåverkan för varje riskobjekt. Faktorer som påverkar affärsrisken inkluderar följande:

- Användningsfrekvensen av den berörda funktionen
- Affärsförlust
- Ekonomisk skada
- Ekologiska eller sociala förluster eller ansvar
- Civil- eller straffrättsliga sanktioner
- Frågor kring funktionssäkerhet
- Böter, förlust av licens
- Brist på rimliga lösningar om människor inte kan arbeta mer
- Funktionalitetens synlighet
- Synlighet av felsymptom som leder till negativ publicitet och potentiell image-skada
- Förlust av kunder

Med tanke på den tillgängliga riskinformationen måste testanalytiker fastställa affärsrisknivåerna enligt de riktlinjer som tillhandahålls av en testansvarig. Dessa kan klassificeras med hjälp av en ordningsskala (numeriskt eller låg/ medium/hög) eller använda trafiksignalsfärger. När riskens sannolikhet och konsekvens har tilldelats använder testansvariga dessa värden för att bedöma risknivån för varje risk. Risknivån används därefter för att prioritera de riskreducerande aktiviteterna [VanVeenendaal12].

2.4 Riskreducering

Under projektet bör testanalytiker försöka göra följande:

- Minska produktrisken genom att designa effektiva testfall som otvetydigt visar om testet godkänns eller underkänts, och genom att delta i granskningar av programvaruprodukter som krav-, design- och användardokumentation
- Implementera lämpliga riskreducerande aktiviteter identifierade i teststrategin och testplanen (till exempel testa en affärsprocess med särskilt hög risk med särskilda testtekniker)
- Omvärdera kända risker baserat på ytterligare information som samlas in när projektet utvecklas, justera risksannolikhet, riskkonsekvens eller båda, efter behov
- Identifiera nya risker efter information som erhållits under testningen

När man pratar om en produktrisk är testning ett väsentligt bidrag till att mildra sådana risker. Genom att hitta defekter minskar testarna risken genom att upplysa om defekterna och möjligheterna att hantera dessa före release. Om testarna inte hittar några defekter så har testningen reducerat risken

genom att ge bevis på att systemet, under vissa förhållanden (dvs. de testade villkoren), fungerar korrekt. Testanalytiker hjälper till att bestämma alternativ för riskreducering genom att undersöka möjligheterna att bland annat samla in korrekt testdata, skapa och testa realistiska användarscenarier och genomföra eller övervaka användbarhetsstudier.

2.4.1 Prioritering av testerna

Riskenivån används också för att prioritera testerna. En testanalytiker kan bestämma att det finns en hög risk inom transaktionsnoggrannheten i ett redovisningssystem. Som ett resultat, för att mildra risken, kan testaren samarbeta med andra experter på affärsområdet för att samla in en stark uppsättning av testdata som kan behandlas och verifieras för noggrannheten. På liknande sätt kan en testanalytiker bestämma att användbarhetsproblem är en betydande risk för ett nytt testobjekt. I stället för att vänta på att ett användaracceptanstest upptäcker några problem, kan testanalytikern prioritera ett tidigt användbarhetstest baserat på en prototyp för att hjälpa till att identifiera och lösa användbarhetsdesignproblem tidigt innan användaracceptantestet. Denna prioritering måste övervägas så tidigt som möjligt i planeringsstadierna så att tidsplanen kan rymma den nödvändiga testningen vid den nödvändiga tidpunkten.

I vissa fall exekveras de alla högsta risktesterna före tester med lägre risk, och testerna genomförs i strikt riskordning (kallat "depth-first"). I andra fall används en urvalsmetod för att välja ett urval av tester över alla identifierade riskområden med hjälp av riskenivå för att väga urvalet och samtidigt säkerställa täckning av alla risker minst en gång (kallat "breadth-first").

Oavsett om riskbaserad testning går depth-first eller breadth-frist är det möjligt att den tid som avsatts för testning kan ha runnit ut utan att alla tester har exekverats. Riskbaserad testning gör det möjligt för testare att rapportera till ledningen när det gäller den återstående riskenivån vid denna punkt, och överläter till ledningen att besluta om de ska utvidga testningen eller överföra återstående risk till användare, kunder, helpdesk/teknisk support och/eller operativ personal.

2.4.2 Anpassa testningen för framtida testcykler

Riskbedömning är inte en engångsaktivitet som utförs innan testimplementering påbörjas; det är en kontinuerlig process. Varje framtida planerad testcykel bör underkastas ny riskanalys för att ta hänsyn till sådana faktorer som:

- Några nya eller betydligt förändrade produktrisker
- Ostabila eller felbenägna områden som upptäcktes under testningen
- Risker från åtgärdade defekter
- Typiska defekter som upptäcks under testningen
- Undertestade områden (låg kravtäckningsgrad)

3. Testtekniker - 630 minuter.

Nyckelord

användningsfallsbaserad testning, black-box-testteknik, checklistebaserad testning, defektbaserad testteknik, defekttaxonomi, ekvivalensklassindelning, erfarenhetsbaserad testning, erfarenhetsbaserad testteknik, felgissning, gränsvärdesanalys, klassifikationsträdsteknik, parvis testning, testcharter, testning med hjälp av beslutstabeller, tillståndsbaserad testning, utforskande testning

Utbildningsmål för testtekniker

3.1 Introduktion

Inga utbildningsmål

3.2 Black-box-testtekniker

- TA-3.2.1 (K4) Analysera ett visst specifikationsobjekt och designtestfall genom att tillämpa ekvivalensklassindelning
- TA-3.2.2 (K4) Analysera ett visst specifikationsobjekt och designtestfall genom att tillämpa gränsvärdesanalys
- TA-3.2.3 (K4) Analysera ett visst specifikationsobjekt och designtestfall genom att tillämpa testning med hjälp av beslutstabeller
- TA-3.2.4 (K4) Analysera ett visst specifikationsobjekt och designtestfall genom att tillämpa tillståndsbaserad testning
- TA-3.2.5 (K2) Förklara hur klassifikationsträddiagram hjälper testtekniker
- TA-3.2.6 (K4) Analysera ett visst specifikationsobjekt och designtestfall genom att tillämpa parvis testning
- TA-3.2.7 (K4) Analysera ett visst specifikationsobjekt och designtestfall genom att tillämpa användningsfallsbaserad testning
- TA-3.2.8 (K4) Analysera ett system, eller dess kravspecifikation, för att bestämma vilka troliga typer av defekter som ska hittas och välja lämplig(a) black-box-testteknik(er)

3.3 Erfarenhetsbaserad testteknik

- TA-3.3.1 (K2) Förklara principerna för erfarenhetsbaserade testtekniker och fördelarna och nackdelarna jämfört med black-box och defektbaserade testtekniker
- TA-3.3.2 (K3) Identifiera utforskande tester från ett givet scenario
- TA-3.3.3 (K2) Beskriv tillämpningen av defektbaserade testtekniker och differentiera deras användning från testtekniker i black-box-testtekniker

3.4 Tillämpa de mest lämpliga testteknikerna

- TA-3.4.1 (K4) För en viss projektsituation, bestäm vilken black-box- eller erfarenhetsbaserad testteknik som ska användas för att uppnå specifika mål

3.1 Introduktion

Testteknikerna som behandlas i detta kapitel är indelade i följande kategorier:

- Black-box
- Erfarenhetsbaserade

Dessa tekniker är komplementära och kan lämpligen användas för varje given testaktivitet, oavsett vilken testnivå som det gäller.

Observera att båda kategorierna av tekniker kan användas för att testa funktionella och icke-funktionella kvalitetsegenskaper. Testning av programvaruegenskaper behandlas i nästa kapitel.

Testteknikerna som behandlas i dessa avsnitt kan huvudsakligen fokusera på att bestämma optimala testdata (till exempel från ekvivalensklasser) eller härleda testprocedurer (till exempel från tillståndsmodeller). Det är vanligt att kombinera tekniker för att skapa kompletta testfall.

3.2 Black-Box-testtekniker

Black-box-testtekniker introducerades i ISTQB® Foundation Level- kursplanen [ISTQB_FL_SYL].

Vanliga funktioner för black-box-testtekniker inkluderar:

- Modeller, till exempel tillståndsovergångsdiagram och beslutstabeller, skapas under testdesign enligt testtekniken
- Testvillkor härleds systematiskt från dessa modeller

Testtekniker ger i allmänhet täckningskriterier, som kan användas för att mäta testdesign- och testexekveringsaktiviteter. Att fullständigt uppfylla täckningskriterierna betyder inte att hela uppsättningen tester måste vara fullständig, utan snarare att modellen inte längre föreslår några ytterligare tester för att öka täckningen baserat på den tekniken.

Black-box-testning är vanligtvis baserad på någon form av specifikationsdokumentation, till exempel en systemkravspecifikation eller användarberättelser. Eftersom specifikationsdokumentationen ska beskriva systembeteende, särskilt när det gäller funktionell lämplighet, är härledning av tester från kraven ofta en del av att testa systemets beteende. I vissa fall kan det inte finnas någon specifikationsdokumentation, men det finns underförstådda krav, till exempel att ersätta den funktionella lämpligheten hos ett tidigare system.

Det finns ett antal black-box-testtekniker. Dessa tekniker är inriktade på olika typer av programvara och scenarier. Kapitlet nedan visar användbarheten för varje teknik, vissa begränsningar och svårigheter som testanalytikern kan uppleva, metoden med vilken täckning mäts och vilka typer av defekter som man fokuserar på.

Se [ISO29119-4], [Bath14], [Beizer95], [Black07], [Black09], [Copeland04], [Craig02], [Forgács19], [Koomen06] och [Myers11] för mer information.

3.2.1 Ekvivalensklassindelning

Ekvivalensklassindelning (EP) är en teknik som används för att minska antalet testfall som krävs för att effektivt testa hanteringen av in och utdata, interna värden och tidsrelaterade värden.

Ekvivalensklassindelning (equivalence partitioning) används för att skapa ekvivalensklasser från uppsättningar av värden som måste behandlas på samma sätt. Genom att välja ett representativt värde från en klass antas att täckning för alla objekt i samma klass är uppfyllt.

Vanligtvis bestämmer flera parametrar testobjektets beteende. När ekvivalensklasser för olika parametrar kombineras till testfall kan olika tekniker tillämpas.

Tillämpning

Denna teknik är tillämpbar på vilken testnivå som helst och är lämplig när alla individer i en uppsättning av värden som ska testas förväntas hanteras på samma sätt och där de uppsättningar av värden som används av applikationen inte interagerar. En ekvivalensklass kan vara vilken icke-tom uppsättning av värden som helst, till exempel: ordnad, oordnad, diskret, kontinuerlig, oändlig, ändlig eller till och med en enskild. Valet av uppsättningar av värden är tillämpligt på giltiga och ogiltiga klasser (dvs. klasser som innehåller värden som bör anses ogiltiga för den programvara som testas).

Ekvivalensklassindelning är starkast när den används i kombination med gränsvärdesanalys som utvidgar testvärdena till att omfatta gränserna hos klasserna. Ekvivalensklassindelning, som använder värden från giltiga klasser, är en vanligt använd teknik för smoke testning av ett nytt bygge eller en ny release eftersom det snabbt avgör om grundläggande funktioner fungerar.

Begränsningar/Svårigheter

Om antagandet är felaktigt och värdena i klassen inte hanteras på exakt samma sätt kan den här tekniken missa defekter. Det är också viktigt att välja klasserna noggrant. Till exempel kan ett inmatningsfält som accepterar positiva och negativa siffror testas bättre som två giltiga klasser, en för de positiva siffrorna och ett för de negativa siffrorna, på grund av sannolikheten för olika hantering. Beroende på om noll tillåts eller inte, kan detta bli en annan klass. Det är viktigt för en testanalytiker att förstå den underliggande bearbetningen för att bestämma den bästa uppdelningen av värdena. Detta kan kräva stöd för att förstå koddesign.

Testanalytikern bör även beakta tänkbara beroenden mellan olika parametrars ekvivalensklasser. I ett flygbokningssystem får exempelvis parametern "medföljande vuxen" endast användas i kombination med åldersklassen "barn".

Täckning

Täckningen bestäms genom att ta antalet klasser för vilka ett värde har testats och dela det antalet med antalet klasser som har identifierats. Ekvivalensklassindelningstäckning anges sedan i procent. Att använda flera värden för en enda klass ökar inte täckningsgraden.

Om testobjektets beteende beror på en enskild parameter bör varje ekvivalensklass täckas minst en gång, oavsett om den är giltig eller ogiltig.

Om det handlar om mer än en parameter bör testanalytikern välja en enkel eller kombinatorisk täckningstyp beroende på risken [Offutt16]. Därför är det viktigt att skilja på kombinationer som enbart innehåller giltiga klasser och kombinationer från de som innehåller en eller flera ogiltiga klasser. För kombinationer som enbart innehåller giltiga ekvivalensklasser är minimikravet en enkel täckning av alla giltiga klasser som sträcker sig över alla parametrar. Det minsta antalet testfall som behövs i en sådan testsvit motsvarar det högsta antalet giltiga klasser hos en parameter, förutsatt att parametrarna är oberoende av varandra. Täckningstyper som är mer djupgående och relaterade till kombinatoriska tekniker är den parvisa täckningen (se 3.2.6 nedan) eller den kompletta täckningen av kombinationer av alla giltiga klasser. Ogiltiga ekvivalensklasser bör testas åtminstone individuellt, till exempel i kombination med giltiga klasser för övriga parametrar, för att undvika defektmaskering. Så varje ogiltig klass bidrar med ett testfall till en testsvit för enkel täckning. Vid hög risk kan ytterligare kombinationer läggas till i testsviten, exempelvis endast ogiltiga klasser eller par av ogiltiga klasser.

Typ av defekter

En testanalytiker använder denna teknik för att hitta defekter vid hanteringen av olika datavärden.

3.2.2 Gränsvärdesanalys

Gränsvärdesanalys (Boundary Value Analysis, BVA) används för att testa korrekt hantering av värden som finns på gränserna hos ordnade ekvivalensklasser. Två tillvägagångssätt för gränsvärdesanalys är vanligast: testning av två gränsvärden och testning av tre gränsvärden. Vid tvåvärdestestning används gränsvärdet (på gränsen) och värdet som ligger precis utanför gränsen (med minsta möjliga precision utifrån erforderlig noggrannhet). Om till exempel summor i en valuta har två decimaler och klassen inkluderar värden från 1 till 10, skulle tvåvärdestesternas värden för den övre gränsen vara 10 och 10,01. De nedre gränstestvärdena skulle vara 1 och 0,99. Gränserna definieras av max- och minvärdena i den definierade ekvivalensklassen.

För testning med tre gränsvärden används värdena före, på och över gränsen. I föregående exempel skulle de övre gränstesterna inkludera 9,99, 10 och 10,01. De nedre gränstesterna skulle inkludera 0,99, 1 och 1,01. Beslutet om att använda två- eller trevärdes gränsvärdestestning bör baseras på den risk som är förknippad med det objekt som testas, varvid metoden med tre gränsvärden används för objekt med högre risk.

Tillämpning

Denna teknik är tillämpbar på alla testnivåer och är lämplig när det finns ekvivalensklasser i ordning. Därför utförs gränsvärdesanalystekniken ofta tillsammans med ekvivalensklassindelningstekniken. Ekvivalensklasser i ordning krävs för att begreppet att vara innanför och utanför gränsen ska fungera. Till exempel är ett intervall av nummer en ordnad klass. En klass som består av textsträngar kan exempelvis vara alfabetiskt ordnad, men om ordningen inte är relevant ur affärssynpunkt så ska gränsvärdesanalys inte vara i fokus. Förutom numeriska värden kan gränsvärdesanalys tillämpas på följande:

- Numeriska attribut för icke-numeriska variabler (till exempel längd)
- Antal loopar av exekveringscykler, inkluderande loopar i tillståndsdigram
- Antal itererande element i lagrade datastrukturer såsom matriser
- Storleken på fysiska objekt, till exempel minne
- Varaktighet hos aktiviteter

Begränsningar/Svårigheter

Eftersom noggrannheten hos denna teknik beror på den exakta identifieringen av ekvivalensklasser för att korrekt identifiera gränserna, är den utsatt för samma begränsningar och svårigheter som i ekvivalensklassindelning. Testanalytikern bör också vara medveten om precisionen i de giltiga och ogiltiga värdena för att kunna bestämma exakt de värden som ska testas. Endast klasser i ordning kan användas för gränsvärdesanalys, men detta är inte begränsat till ett antal giltiga iindata. Till exempel, när testning för antalet celler som stöds av ett kalkylark, finns det en klass som innehåller antalet celler upp till och inklusive maximalt tillåtna celler (gränsen) och en annan klass som börjar med en cell över det maximala (över gränsen).

Täckning

Täckning bestäms genom att ta antalet gränsvillkor som testas och dela det med antalet identifierade gränsvillkor (antingen med hjälp av två- eller trevärdesmetoden). Täckningen anges i procent.

Typ av defekter

Gränsvärdesanalys kan säkra upptäckten av förskjutna eller missade gränser och kan hitta fall med extra gränser. Denna teknik hittar defekter gällande hanteringen av gränsvärdena, särskilt fel med mindre än- och större än-logik (dvs. förskjutning). Det kan också användas för att hitta icke-funktionella defekter, till exempel att ett system stödjer 10 000 användare samtidigt, men inte 10 001.

3.2.3 Testning med hjälp av beslutstabeller

En beslutstabell är en tabulär representation av villkor och relaterade åtgärder och uttryckta regler som indikerar vilka åtgärder som hör till vilka villkor [OMG-DMN]. Testanalytikern kan använda beslutstabellerna för att analysera de regler som gäller för den programvara som testas samt designa tester som täcker dessa regler.

Testobjektets villkor och resulterande åtgärder formar beslutstabellens rader, vanligtvis med villkoren högst upp och åtgärderna längst ner. Tabellens första kolumn innehåller beskrivningar av villkoren respektive åtgärderna. Följande kolumner, så kallade regler, innehåller villkorsvärden respektive motsvarande åtgärdsvärden.

Beslutstabeller där villkoren är booleska med enkla värden som "Sant" och "Falskt" kallas beslutstabell med begränsade villkor (limited-entry). Ett exempel på ett sådant villkor är "Användarens inkomst < 1000". Beslutstabeller med utökade villkor (extended-entry) gör det möjligt för villkoren att ha flera värden som kan representera diskreta element eller elementuppsättningar. Exempelvis kan ett villkor "Användarens inkomst" anta ett av tre möjliga värden: "lägre än 1000", "mellan 1000 och 2000" och "mer än 2000".

Enkla åtgärder antar booleska värdena "Sant" och "Falskt" (till exempel kan åtgärden "Tilldelad rabatt = 20%" anta värdena "Sant", som betecknas "X" om åtgärden inträffar, och "Falskt", som betecknas "-", om den inte inträffar). Precis som med villkor tar åtgärder värden från andra domäner. Exempelvis kan åtgärden "Tilldelad rabatt" ta ett av fem möjliga värden: 0%, 10%, 20%, 35% och 50%.

Testning av beslutstabeller startar med att designa beslutstabeller baserat på specifikationen. Regler som innehåller omöjliga kombinationer av villkorsvärden exkluderas eller markeras som "omöjliga". Vidare bör testanalytikern granska beslutstabellerna tillsammans med övriga intressenter. Testanalytikern bör försäkra sig om att tabellens regler är konsekventa (d.v.s. att reglerna inte överlappar varandra), kompletta, (d.v.s. att de innehåller en regel för varje möjlig villkorsvärdeskombination), och korrekta (d.v.s. att de modellerar det avsedda beteendet).

Den grundläggande principen för beslutstabellstestning är att reglerna formar testvillkoren.

När man designar ett testfall som ska täcka en given regel bör testanalytikern vara medveten om att testfallens ingångsvärden kan skilja sig från beslutstabellens villkorsvärden. Exempelvis behöver värdet "SANT" för villkoret "ärlig inkomst > 100,000?" inte vara direkt tillämpligt utan kan kräva arbete från testaren för att definiera en klients konto med krediter som överstiger 100,000 under ett givet räkenskapsår. Således kan de förväntade resultaten från testfallet skilja sig från beslutstabellens åtgärder.

När beslutstabellen har färdigställts behöver reglerna implementeras som testfall genom att välja testfallens ingångsvärden (och de förväntade resultaten) som uppfyller villkoren och åtgärderna.

Komprimerade beslutstabeller

När man försöker testa alla möjliga ingångskombinationer i enlighet med villkoren, kan beslutstabellerna bli mycket stora. En komplett beslutstabell med begränsade ingångar (limited-entry) och n villkor har 2^n regler. En teknik som används för att systematiskt minska antalet kombinationer kallas testning med komprimerade beslutstabeller (collapsed decision table testing) [Mosley93]. När denna teknik används kan en grupp regler med samma uppsättning av åtgärder reduceras (komprimeras) till en regel, om det inom denna grupp finns villkor som inte är relevanta för åtgärden och om alla andra villkor förblir oförändrade. I denna resulterande regel betecknas irrelevanta

villkorsvärden med "utelämna", vilka vanligen markeras med bindestreck "-". För villkor med "utelämna"-värden kan testanalytikern specificera slumpmässigt giltiga värden för testimplementering.

För kollapsade regler finns fall där villkorsvärdet inte är applicerbart i kombination med ett annat villkorsvärde eller när två eller fler villkor har motstridiga värden. Till exempel, i en beslutstabell för kortbetalningar är villkoret "PIN-koden är korrekt" inte applicerbart om villkoret för "godkänt kort" är falskt.

Komprimerade beslutstabeller kan ha färre regler än de kompletta beslutstabellerna, vilket resulterar i ett lägre antal testfall och mindre ansträngning. Om en given regel har poster med "utelämna" och enbart ett testfall som täcker denna regel, kommer endast ett av flera möjliga värden att testas för den regeln, så att en defekt som involverar andra värden kan förbli upptäckta. Därför bör testanalytikern till följd av de höga riskerna och i enlighet med testledaren, definiera separata regler för varje möjlig kombination av de enskilda villkorsvärdena istället för att komprimera beslutstabellen.

Tillämpning

Beslutstabelltestning tillämpas vanligtvis på integration-, system- och acceptanstestnivåerna. Den kan också användas vid komponenttestning när en komponent innehåller en uppsättning beslutslogik. Denna teknik är särskilt användbar när testobjektet är specificerat i form av flödesscheman eller tabeller med verksamhetsregler.

Beslutstabeller är också en kravdefinitionsteknik och kravspecifikationer kan ibland redan ha definierats i detta format. Testanalytikern bör ändå delta i granskningen av beslutstabeller och analysera de innan testdesign påbörjas.

Begränsningar/Svårigheter

När kombinationer av villkor övervägs kan det vara en utmaning att hitta alla interagerande villkor, särskilt när kraven inte är väl definierade eller inte finns. Man måste vara försiktig när man väljer villkor från en beslutstabell så att antalet kombinationer av dessa villkor förblir hanterbara. I värsta fall kommer antalet regler att öka exponentiellt.

Täckning

Den vanliga täckningsstandarden för denna teknik är att täcka samtliga regler i beslutstabeller i ett testfall. Täckningen mäts som ett procentuellt antal regler täckta av en testsvit jämfört med det totala antalet möjliga regler.

Gränsvärdesanalys och ekvivalensklassindelning kan kombineras med beslutstabellstekniken, speciellt med beslutstabeller med utökade villkor. Om villkoren innehåller ekvivalensklassindelning som är totalt ordnade kan gränsvärden användas som ytterligare villkor, vilket i sin tur leder till ytterligare regler och testfall.

Typ av defekter

Typiska defekter inkluderar felaktig logik-relaterad bearbetning baserat på speciella kombinationer av villkor som resulterar i oväntade resultat. Under skapandet av beslutstabellerna kan defekter hittas i specifikationsdokumenten. Det är inte ovanligt att förbereda en uppsättning villkor och besluta att det förväntade resultatet är ospecificerat för en eller flera regler. De vanligaste typerna av defekter är saknade åtgärder (dvs. det finns ingen information om vad som egentligen bör hända i en viss situation) och motsägelser.

3.2.4 Tillståndsbaserad testning

Tillståndsbaserad testning används för att testa testobjektets förmåga att göra övergångar till och från definierade tillstånd via giltiga övergångar, såväl som att försöka komma till ogiltiga tillstånd eller täcka ogiltiga övergångar. Händelser får testobjektet att övergå från tillstånd till tillstånd och exekvera

handlingar. Händelser kan kvalificeras av villkor (ibland kallade "guard conditions" eller "transition guards") som påverkar vilken övergång som ska tas. Till exempel kommer en inloggningshändelse med giltigt användarnamn/lösenord att resultera i en annan övergång än en inloggningshändelse med ett ogiltigt lösenord. Denna information representeras i ett tillståndsdigram eller i en tillståndstabell (som också kan innehålla eventuella ogiltiga övergångar mellan tillstånd).

Tillämpning

Tillståndsbaserad testning är tillämplig för all programvara som har definierade tillstånd och har händelser som kommer att orsaka övergångar mellan dessa tillstånd (till exempel ändra skärmbild). Tillståndsbaserad testning kan användas på vilken testnivå som helst. Inbäddad programvara, webbprogramvara och alla typer av transaktionsprogramvara är bra kandidater för denna typ av testning. Även styrsystem, till exempel trafikljuskontroller, är också bra kandidater för denna typ av testning.

Begränsningar/Svårigheter

Att bestämma tillstånden är ofta den svåraste delen av att definiera tillståndsdigrammet eller tillståndstabellen. När testobjektet har ett användargränssnitt används ofta de olika skärmbilderna som visas för användaren för att representera tillstånden. För inbäddad programvara kan tillstånden vara beroende av hårdvarans tillstånd.

Förutom själva tillstånden är den grundläggande enheten för tillståndsbaserad testning den individuella övergången. Att helt enkelt testa alla enstaka övergångar hittar vissa typer av tillståndsövergångsdefekter, men mer kan hittas genom att testa sekvenser av övergångar. En enskild övergång kallas 0-switch: en sekvens av två på varandra följande övergångar kallas 1-switch; en sekvens av tre på varandra följande övergångar kallas 2-switch, och så vidare. I allmänhet så representerar en N-switch $N+1$ succesiva övergångar [Chow1978]. När N ökar så ökar antalet N-switchar snabbt, vilket gör det svårt att uppnå en täckning av N-switchar med ett rimligt antal tester.

Täckning

Som med andra typer av testtekniker finns det en hierarki av täckningsnivåer. Minsta acceptabla täckningsgrad är att ha besökt varje tillstånd och korsat varje övergång minst en gång. 100% övergångstäckning (även känd som 100% 0-switch-täckning) kommer att garantera att varje tillstånd besöks och varje övergång korsas, såvida inte systemdesignen eller tillståndsövergångsmodellen (diagram eller tabell) är defekt. Beroende på förhållandena mellan tillstånd och övergångar kan det vara nödvändigt att korsa vissa övergångar mer än en gång för att utföra andra övergångar en gång.

Uttrycket "N-switch-täckning" avser antalet switchar täckta med längden $N+1$, i procent av det totala antalet switchar av den längden. För att exempelvis uppnå 100% 1-switch-täckning krävs det att varje giltig sekvens av två på varandra följande övergångar har testats minst en gång. Denna testning kan utlösa vissa typer av felsymptom som 100% 0-switch-täckning skulle missa.

"Round-trip täckning" gäller situationer där övergångssekvenser bildar loopar. 100% round-trip täckning uppnås när alla loopar från något tillstånd tillbaka till samma tillstånd har testats för alla tillstånd där looparna börjar och slutar. Denna loop kan inte innehålla mer än en förekomst av något speciellt tillstånd (utom det initiala/sista) [Offutt16].

För någon av dessa tillvägagångssätt kommer en ännu högre täckningsgrad att inkludera alla ogiltiga övergångar som identifieras i en tillståndstabell. Täckningskrav och täcknings-set för tillståndsbaserad testning måste identifiera om ogiltiga övergångar ingår.

Testfallsdesignen för att uppnå önskad täckning stöds av tillståndsdigrammet eller tillståndstabellen för det specifika testobjektet. Denna information kan också representeras i en tabell som visar N-switch-övergångarna för ett visst värde på "N" [Black09].

En manuell procedur kan tillämpas för att identifiera element som ska täckas (till exempel övergångar, tillstånd eller N-switchar). En föreslagen metod är att skriva ut tillståndsdigrammet och tillståndstabellen och använda en penna för att markera de täckta objekten tills önskad täckning visas [Black09]. Detta tillvägagångssätt skulle bli för tidskrävande för mer komplexa tillståndsdigram och tillståndstabeller. Ett verktyg bör därför användas för att stödja tillståndsbaserad testning.

Typ av defekter

Typiska defekter inkluderar följande (se även [Beizer95]):

- Felaktiga händelsetyper eller värden
- Felaktiga handlingstyper eller värden
- Felaktigt initialtillstånd
- Oförmåga att nå ett visst/vissa utgångstillstånd
- Oförmåga att komma in önskat tillstånd
- Extra (onödiga) tillstånd
- Oförmåga att exekvera en viss/vissa giltiga övergångar korrekt
- Möjlighet att exekvera ogiltiga övergångar
- Felaktiga "guard conditions"

Under skapandet av en tillståndsövergångsmodell kan defekter upptäckas i specifikationsdokumentet. De vanligaste typerna av fel är utelämnanden (det finns ingen information om vad som egentligen bör hända i en viss situation) och motsägelser.

3.2.5 Klassifikationsträdsteknik

Klassifikationsträd stöder vissa black-box-testtekniker genom att möjliggöra en grafisk representation av det datautrymme som gäller för testobjektet.

Data är organiserat i klassificeringar och klasser enligt följande:

- Klassificeringar: Dessa representerar parametrar i datautrymmet för testobjektet såsom inparametrar (som vidare kan innehålla miljötillstånd och förutsättningar), och utparametrar. Om till exempel en applikation kan konfigureras på många olika sätt kan klassificeringarna inkludera klient, webbläsare, språk och operativsystem.
- Klasser: Varje klassificering kan ha vilket antal klasser och underklasser som helst som beskriver förekomsten av parametern. Varje klass, eller ekvivalensklassindelning, är ett specifikt värde inom en klassificering. I exemplet ovan kan språkklassificeringen innehålla ekvivalensindelningar för engelska, franska och spanska.

Klassifikationsträd gör det möjligt för testanalytikern att ange kombinationer efter behov. Detta inkluderar till exempel parvisa kombinationer (se Kapitel 3.2.6), three-wise-kombinationer och single-wise.

Ytterligare information om användning av klassifikationsträdstekniken finns i [Bath14] och [Black09].

Tillämpning

Skapandet av ett klassifikationsträd hjälper en testanalytiker att identifiera parametrar och deras ekvivalensindelningar och (klasser) som är av intresse.

Ytterligare analys av klassifikationsträdsdiagrammet gör det möjligt att identifiera möjliga gränsvärden och vissa kombinationer av indata som antingen är av särskilt intresse eller kan bortses från (till exempel eftersom de inte är kompatibla). Det resulterande klassifikationsträdet kan sedan användas för att stödja ekvivalensklassindelning, gränsvärdesanalys eller parvis testning (se Kapitel 3.2.6).

Begränsningar/Svårigheter

När mängden klassificeringar och/eller klasser ökar blir diagrammet större och användningen svårare. Klassifikationsträdstechniken skapar inte heller kompletta testfall, endast testdatakombinationer. Testanalytikern måste tillföra resultat för varje testkombination för att skapa kompletta testfall.

Täckning

Testfall kan utformas för att till exempel uppnå minimiklasstäckning (dvs. alla värden i en klassificering testas minst en gång). Testanalytikern kan också besluta att täcka parvisa kombinationer eller använda andra typer av kombinatorisk testning, som till exempel three-wise.

Typ av defekter

Vilka typer av defekter som hittas beror på den teknik(er) som klassifikationsträden stödjer (dvs. ekvivalensklassindelning, gränsvärdesanalys eller parvis testning).

3.2.6 Parvis testning

Parvis testning används vid testning av programvara där flera indataparametrar, var och en med flera möjliga värden, måste testas i kombination, vilket ger upphov till fler kombinationer än vad som är möjligt att testa under den tillåtna tiden. Inparametrarna kan vara oberoende i den meningen att vilket alternativ som helst för vilken faktor som helst (dvs. vilket värde som helst för någon inparameter), dock är detta inte alltid fallet (se notering för presenterade modeller nedan). Kombinationen av en specifik parameter (variabel eller faktor) med ett specifikt värde för den parametern kallas ett parametervärdepar (till exempel om "färg" är en parameter med sju tillåtna värden inklusive "röd", då skulle "färg = röd" kunna vara ett parametervärdepar).

Parvis testning använder kombinatoriska tekniker för att säkerställa att varje parametervärdepar testas en gång mot varje parametervärdepar i varje enskild parameter (dvs. "alla par" av två olika parametervärdepar testas), samtidigt som man undviker att testa alla kombinationer av parametervärdepar. Testanalytikern kan då använda ett manuellt arbetssätt med en tabell där testfall representeras av rader och med en kolumn för varje parameter. Testanalytikern fyller då tabellen med värden så att alla parvärden kan identifieras i tabellen (se [Black09]). Alla poster i tabellen som lämnats tomma kan fyllas med värden av testanalytikern med hjälp av sin egen domänkunskap.

Det finns ett antal verktyg tillgängliga för att hjälpa testanalytikern med denna uppgift (se till exempel www.pairwise.org). De kräver som indata en lista över parametrarna och deras värden och genererar sedan en lämplig uppsättning av värdekombinationer från varje parameter som täcker alla par av parametervärdepar. Output från verktyget kan användas som indata för testfall. Observera att testanalytikern måste ge de förväntade resultaten för varje kombination som skapas av verktygen.

Klassifikationsträd (se Kapitel 3.2.5) används ofta i samband med parvis testning [Bath14]. Klassifikationsträddesign stöds av verktyg och gör det möjligt att visualisera kombinationer av parametrar och deras värden (vissa verktyg erbjuder parvis förbättring). Detta hjälper till att identifiera följande information:

- Indata som ska användas med den parvisa testtekniken.
- Speciella kombinationer av intresse (till exempel ofta använda eller en vanlig orsak till defekter)
- Särskilda kombinationer som är inkompatibla. Detta förutsätter inte att de kombinerade faktorerna inte påverkar varandra; det kan de mycket väl, men borde påverka varandra på acceptabla sätt.
- Logiska samband mellan variabler. Till exempel, "om variabel 1 = x, kan variabel 2 inte vara y". Klassifikationsträd som identifierar dessa förhållanden kallas "feature models".

Tillämpning

Problemet med att ha för många kombinationer av parametervärden visar sig i minst två olika situationer relaterade till testning. Vissa testobjekt involverar flera parametrar vardera med ett antal möjliga värden, till exempel en skärmbild med flera inmatningsfält. I detta fall utgör kombinationer av parametervärden indata för testfallen. Dessutom kan vissa system vara konfigurerbara i ett antal dimensioner, vilket resulterar i ett potentiellt stort konfigurationsutrymme. I båda dessa situationer kan parvis testning användas för att identifiera en delmängd av kombinationer som är hanterbara och möjliga.

För parametrar med många värden kan ekvivalensklassindelning, eller någon annan valmekanism först tillämpas på varje individuell parameter för att minska antalet värden för varje parameter, innan parvis testning tillämpas för att minska uppsättningen resulterande kombinationer. Att identifiera parametrarna och deras värden i ett klassifikationsträd stöder denna aktivitet.

Dessa tekniker används vanligtvis för komponentintegrations-, system- och systemintegrationstestnivåer.

Begränsningar/Svårigheter

Den största begränsningen med dessa tekniker är antagandet att resultaten från några få tester är representativa för alla tester och att dessa få tester representerar den förväntade användningen. Om det finns en oväntad interaktion mellan vissa variabler, kan det förbli upptäckt med den här typen av testtekniker om den speciella kombinationen inte testas. Dessa tekniker kan vara svåra att förklara för en icke-tekniska åhörare eftersom de kanske inte förstår den logiska minskningen av testerna. Alla sådana förklaringar bör balanseras genom att nämna resultaten från empiriska studier [Kuhn16], som visade att inom området för medicintekniska apparater som studerades, triggades 66% av felsymptomen av en enda variabel och 97% av antingen en variabel eller två variabler som interagerade. Det finns en kvarstående risk att parvis testning kanske inte upptäcker systemfel där tre eller flera variabler interagerar.

Identifiering av parametrarna och deras respektive värden är ibland svårt att uppnå. Därför bör detta genomföras med stöd av klassifikationsträd där det är möjligt (se Kapitel 3.2.5). Att hitta en minimal uppsättning kombinationer för att tillfredsställa en viss täckningsnivå är svårt att göra manuellt. Verktyg kan användas för att hitta den minsta möjliga uppsättningen kombinationer. Vissa verktyg stöder möjligheten att tvinga vissa kombinationer att inkluderas i eller uteslutas från det slutliga valet av kombinationer. Testanalytiker kan använda denna möjlighet för att framhäva eller minska faktorer, baserad på sin domänkunskap eller produktanvändningsinformation.

Täckning

100% parvis täckning kräver att varje par av värden av vilket parameterpar som helst ingår i minst en kombination.

Typ av defekter

Den vanligaste typen av defekter som hittas vid denna testteknik är defekter relaterade till de kombinerade värdena för två parametrar.

3.2.7 Användningsfallsbaserad testning

Användningsfallsbaserad testning ger transaktionella, scenariobaserade tester som ska emulera avsedd användning av komponenten eller systemet som specificeras av användningsfallet. Användningsfall definieras i termer av interaktioner mellan aktörerna och en komponent eller system som uppnår något mål. Aktörer kan vara mänskliga användare, extern hårdvara eller andra komponenter eller system.

En vanlig standard för användningsfall finns i [OMG-UML].

Tillämpning

Användningsfallsbaserad testning används vanligtvis inom system- och acceptanstestning. Den kan också användas i integrationstestning där beteendet hos komponenterna eller systemen specificeras av användningsfall. Användningsfall är också ofta grunden för prestandatestning eftersom de visar en realistisk användning av systemet. De scenarier som beskrivs i användningsfallen kan ges till virtuella användare för att skapa en realistisk belastning på systemet (så länge belastnings- och prestandakrav anges i dem eller för dem).

Begränsningar/Svårigheter

För att vara giltiga måste användningsfallet förmedla realistiska användartransaktioner. Användningsfallspecifikationer är en form av systemdesign. Kraven på vad användare behöver utföra bör komma från användare eller användarrepresentanter och bör kontrolleras mot organisatoriska krav innan motsvarande användningsfall designas. Värdet av ett användningsfall reduceras om det inte återspeglar verkliga användar- och organisationskrav, eller hindrar snarare än hjälper till att slutföra användaruppgifter.

En exakt definition av undantags-, alternativ- och felhanteringsbeteenden är viktigt för att täckningen ska vara grundlig. Användningsfall bör tas som riktlinje, men inte en fullständig definition av vad som ska testas eftersom de kanske inte ger en tydlig definition av hela uppsättningen av krav. Det kan också vara fördelaktigt att skapa andra modeller, såsom flödesscheman och/eller beslutstabeller, från användarberättelser för att förbättra testets noggrannhet och för att verifiera själva användningsfallet. Liksom med andra typer av specifikationer kommer detta sannolikt att avslöja logiska avvikelser i användningsfallspecifikationen, om de finns.

Täckning

Den minsta acceptabla täckningsnivån för ett användningsfall är att ha ett testfall för det grundläggande beteendet och tillräckligt med ytterligare testfall för att täcka varje alternativ och felhanteringsbeteende. Om en minimal testsvit krävs kan flera alternativa beteenden bli inkorporerad i ett testfall förutsatt att de är ömsesidigt kompatibla. Om bättre diagnostisk förmåga krävs (till exempel för att hjälpa till att isolera defekter), kan ytterligare ett testfall per alternativt beteende designas, även om kapslade alternativa beteenden fortfarande kommer att kräva att vissa av de beteendena samlas i ett enskilda testfall (till exempel terminerade mot icke-terminerade alternativa beteenden inom ett "retry"-undantagsbeteende).

Typ av defekter

Defekter inkluderar fel hantering av definierat beteende, missade alternativa beteenden, felaktig bearbetning av de presenterade villkoren och dåligt implementerade eller felaktiga felmeddelanden.

3.2.8 Kombination av tekniker

Ibland kombineras tekniker för att skapa testfall. Till exempel kan de villkor som identifieras i en beslutstabelle utsättas för ekvivalensklassindelning för att upptäcka flera sätt som ett villkor kan vara uppfyllt. Testfall skulle då täcka inte bara varje kombination av villkor, utan även för de villkor som är klassindelade, bör ytterligare testfall genereras för att täcka ekvivalensklassindelningen. När testanalytikern väljer den särskilda tekniken som ska tillämpas, bör man överväga användbarheten av tekniken, begränsningarna och svårigheterna och målen för testningen vad gäller täckning och defekter som ska upptäckas. Dessa aspekter beskrivs för de enskilda teknikerna som behandlas i detta kapitel. Det kanske inte finns en enda "bästa" teknik för en situation. Kombinerade tekniker ger ofta den mest fullständiga täckningen förutsatt att det finns tillräckligt med tid och skicklighet för att korrekt använda teknikerna.

3.3 Erfarenhetsbaserade testtekniker

Erfarenhetsbaserad testning utnyttjar testarens skicklighet och intuition, tillsammans med deras erfarenhet av liknande applikationer eller tekniklogier för att rikta in sig på tester för att öka upptäckten av defekter. Dessa testtekniker sträcker sig från "snabbtester" där testaren inte har några formellt förplanerade aktiviteter att utföra, från förplanerade testsessioner med hjälp av testcharters till skriptade testsessioner. De är nästan alltid användbara, men har särskilt värde när aspekter som ingår i följande lista med fördel kan uppnås.

Erfarenhetsbaserad testning har följande fördelar:

- Den kan vara ett bra alternativ till mer strukturerade tillvägagångssätt i de fall då systemdokumentation saknas.
- Den kan tillämpas när testtiden är mycket begränsad.
- Det gör det möjligt att använda tillgänglig expertis inom domänen och tekniken som används i testningen. Detta kan inkludera de som inte är involverade i testning, till exempel från affärsanalytiker, kunder eller klienter.
- Den kan ge tidigt feedback till utvecklarna.
- Den hjälper teamet att bli bekant med programvaran samtidigt som den produceras.
- Den är effektiv när driftsfel analyseras.
- Den gör det möjligt att använda en mångfald testtekniker.

Erfarenhetsbaserad testning har följande nackdelar:

- Den kan vara olämplig i system som kräver detaljerad testdokumentation.
- Höga nivåer av repeterbarhet är svårt att uppnå.
- Förmågan att exakt utvärdera täckningen är begränsad.
- Tester är mindre lämpade för efterföljande automatisering.

När man använder reaktiva och heuristiska tillvägagångssätt använder testarna normalt erfarenhetsbaserad testning, vilken är mer reaktivt till händelser än förplanerade teststrategier. Dessutom pågår exekvering och utvärdering samtidigt. Vissa strukturerade metoder för erfarenhetsbaserad testning är inte helt dynamiska, dvs. testerna skapas inte helt samtidigt som testaren exekverar testet. Detta kan till exempel vara fallet där felgissning används för att inrikta sig på specifika aspekter av testobjektet innan testexekvering.

Observera att även om vissa idéer om täckning presenteras för de tekniker som berörts här, har erfarenhetsbaserade testtekniker inga formella täckningskriterier.

3.3.1 Felgissning

När man använder felgissningstekniken använder en testanalytiker sin erfarenhet för att gissa de potentiella fel som kan ha gjorts när koden designades och utvecklades. När de förväntade felen har identifierats bestämmer en testanalytiker sedan de bästa metoderna att använda för att hitta de resulterande defekterna. Till exempel, om en testanalytiker förväntar sig att programvaran kommer att visa felsymptom när ett ogiltigt lösenord anges, kommer tester att exekveras för att ange en mängd olika värden i lösenordsfältet för att verifiera om felet verkligen finns och har resulterat i en defekt som kan ses som en felsymptom när testerna exekveras.

Förutom att användas som en testteknik är felgissning också användbar vid riskanalys för att identifiera potentiella feltillstånd. [Myers11]

Tillämpning

Felgissning görs främst under integrations- och systemtestning, men kan användas på vilken testnivå som helst. Denna teknik används ofta med andra tekniker och hjälper till att bredda omfattningen av

de befintliga testfallen. Felgissning kan också användas effektivt när man testar en ny release av programvaran för att testa för vanliga defekter innan man börjar mer noggrann och skriptad testning.

Begränsningar/Svårigheter

Följande begränsningar och svårigheter gäller för felgissning:

- Täckning är svårt att bedöma och varierar mycket med testanalytikerns förmåga och erfarenhet.
- Den används bäst av en erfaren testare som är bekant med de typer av defekter som vanligtvis introduceras i den typ av kod som testas.
- Den används ofta men är ofta inte dokumenterad och kan därför vara mindre reproducerbar än annan testning.
- Testfall kan dokumenteras men på ett sätt som bara författaren förstår och kan reproducera.

Täckning

När en defekttaxonomi används bestäms täckningen genom att ta antalet testade taxonomiobjekt dividerat med det totala antalet taxonomiobjekt och ange täckning i procent. Utan en defekttaxonomi begränsas täckningen av testarens erfarenhet och kunskap och den tillgängliga tiden. Mängden defekter som hittas i denna teknik kommer att variera beroende på hur väl testaren kan inrikta sig på problematiska områden.

Typ av defekter

Typiska defekter är vanligtvis de som definieras i den särskilda defekttaxonomier eller sådana som kanske inte har hittats i black-box-testning, och därför "gissas" av testanalytikern.

3.3.2 Checklistebaserad testning

När man tillämpar den checklistebaserade testtekniken använder en erfaren testanalytiker en högnivå-generaliserad lista över objekt som ska noteras, kontrolleras eller kommas ihåg, eller en uppsättning regler eller kriterier som ett testobjekt måste verifieras mot. Dessa checklistor är baserade på en uppsättning standarder, erfarenheter och andra överväganden. Exempelvis kan en standardlista för användargränssnitt användas som bas för att testa en applikation. I agil programvaruutveckling kan checklistor byggas utifrån acceptanskriterierna för en användarberättelse.

Tillämpning

Checklistebaserad testning är mest effektiv i projekt med ett erfaret testteam som är bekant med den programvara som testas eller är bekant med det område som omfattas av checklistan (till exempel för att framgångsrikt tillämpa en användargränssnittschecklista kan testanalytikern känna till användargränssnittstestning men inte det specifika systemet som testas). Eftersom checklistor är på hög nivå och tenderar att sakna de detaljerade stegen som vanligtvis finns i testfall och testprocedurer används testarens kunskap för att fylla i luckorna. Genom att ta bort de detaljerade stegen kräver checklistor mindre underhåll och kan tillämpas i flera liknande releaser.

Checklistor är väl lämpade för projekt där programvara släpps och ändras snabbt. Detta hjälper till att minska både förberedelse- och underhållstid för testdokumentation. De kan användas för vilken testnivå som helst och används också för regressionstestning och smoke-testning.

Begränsningar/Svårigheter

Den höga nivån på checklistorna kan påverka reproducerbarheten av testresultaten. Det är möjligt att flera testare tolkar checklistorna på olika sätt och kommer att följa olika tillvägagångssätt för att uppfylla checklistans innehåll. Detta kan orsaka olika testresultat, även om samma checklista används. Det kan också resultera i en bredare täckning men reproducerbarheten offras ibland. Checklistor kan också leda till en övertro på täckningsnivån som uppnås eftersom den faktiska testningen beror på testarens bedömning. Checklistor kan härledas från mer detaljerade testfall eller

listor och tenderar att växa med tiden. Underhåll krävs för att se till att checklistorna täcker de viktiga aspekterna hos programvaran under test.

Täckning

Täckningen kan bestämmas genom att ta antalet checklisteobjekt som är testade dividerat med det totala antalet checklisteobjekt och täckning anges i procent. Täckningen är lika bra som checklistan, men på grund av den höga nivån hos checklistan kommer resultaten att variera beroende på testanalytikern som exekverar checklistan.

Typ av defekter

Typiska defekter som upptäcks med denna teknik orsakar felsymptom som uppstår till följd av variation i data, stegsekvensen eller det allmänna arbetsflödet under testning.

3.3.3 Utforskande testning

Utforskande testning kännetecknas av att testaren samtidigt lär sig om testobjektet och dess defekter, planerar testarbetet som ska utföras, designar och exekverar testerna och rapporterar resultaten. Testaren justerar dynamiskt testmålen under exekvering och förbereder endast lättviktig dokumentation. [Whittaker09]

Tillämpning

Bra utforskande testning är planerad, interaktiv och kreativ. Det kräver lite dokumentation om systemet som ska testas och används ofta i situationer där dokumentationen inte är tillgänglig eller inte är tillräcklig för andra testtekniker. Utforskande testning används ofta som komplement till andra testtekniker och fungerar som en bas för utvecklingen av ytterligare testfall. Utforskande testning används ofta i agil programvaruutveckling för att göra testning av användarberättelser flexibel och snabb med endast minimal dokumentation. Tekniken kan också tillämpas i projekt som använder en sekventiell utvecklingsmodell.

Begränsningar/Svårigheter

Täckningen av utforskande testning kan vara sporadisk och reproducerbarheten för de exekverade testerna kan vara svår. Att använda testcharter för att utse områden som ska täckas i en testsession och att använda tidsramar (time-boxing) för att bestämma tiden som är tillåten för testningen är tekniker som används för att hantera utforskande testning. I slutet av en testsession eller uppsättning av sessioner kan den testansvariga hålla en debriefingsession för att samla testresultaten och besluta om testcharter för nästa testsessioner.

En annan svårighet med utforskande testsessioner är att exakt spåra dem i ett testhanteringssystem. Detta görs ibland genom att skapa testfall som faktiskt är utforskande testsessioner. Detta gör det möjligt att spåra den tid som avsatts för den utforskande testningen och den planerade täckningen med de andra testinsatserna.

Eftersom reproducerbarhet kan vara svår att uppnå med utforskande testning kan detta också orsaka problem när man behöver repetera stegen för att reproducera ett felsymptom. Vissa organisationer använder in- och uppspelningsförmågan hos ett testautomatiseringsverktyg för att registrera stegen som en utforskande testare har tagit. Detta ger en fullständig registrering av alla aktiviteter under den utforskande testsessionen (eller någon annan erfarenhetsbaserad testsession). Att analysera detaljerna för att hitta den faktiska orsaken till ett felsymptom kan vara tråkigt, men det finns åtminstone en lista över alla steg som var inblandade.

Andra verktyg kan användas för att identifiera utforskande testsessioner, men dessa registrerar inte de förväntade resultaten eftersom de inte identifierar GUI-interaktionen. I detta fall måste de förväntade resultaten noteras så att korrekt analys av defekter kan utföras vid behov. I allmänhet

rekommenderas det att anteckningar också tas när man exekverar utforskande testning för att stödja reproducerbarheten där det behövs.

Täckning

Testcharters kan designas för specifika uppgifter, mål och leveranser. Utforskande testsessioner planeras sedan för att uppnå dessa kriterier. Charten kan också identifiera på vad testansträngningen ska fokuseras, vad som ligger i och utanför testsessionens omfattning och vilka resurser som ska användas för att genomföra de planerade testerna. En testsession kan användas för att fokusera på speciella defekttyper och andra potentiellt problematiska områden som kan hanteras utan formaliteten hos skriptad testning.

Typ av defekter

Typiska defekter som upptäcks med utforskande testning är scenariobaserade problem som missades under skriptad funktionell lämplighetstestning, problem som faller mellan funktionella gränser och arbetsflödesrelaterade problem. Prestanda- och säkerhetsproblem upptäcks ibland även under utforskande testning.

3.3.4 Defektbaserade testtekniker

I en defektbaserad testteknik används den sökta defekttypen som grund för testdesign, med tester som härleds systematiskt från vad som är känt om den defekttypen. Till skillnad från black-box-testning som härleder sina tester från testbasen, härleder defektbaserad testning sina tester från listor som fokuserar på defekter. I allmänhet kan listorna vara organiserade i defekttyper, grundorsaker, felsymptom och annan defektrelaterad data. Standardlistor kan gälla för flera typer av programvara och är inte produktspecifika. Att använda dessa listor hjälper till att utnyttja branschstandardkunskap för att härleda de specifika testerna. Genom att följa branschspecifika listor kan mätningar för felhändelser spåras över projekt och till och med över organisationer. De vanligaste fellistorna är de som är organisations- eller projektspecifika och använder sig av specifik expertis och erfarenhet.

Defektsbaserad testning kan också använda listor över identifierade risker och riskscenarier som grund för inriktning av testningen. Denna testteknik tillåter en testanalytiker att inrikta sig på en specifik typ av defekt eller att arbeta systematiskt genom en lista över kända och vanliga defekter av en viss typ. Utifrån denna information skapar testanalytikern testvillkor och testfall som kommer att få felet att visa sig (om det finns).

Tillämpning

Defektbaserad testning kan tillämpas på vilken testnivå som helst, men oftast under systemtestning.

Begränsningar/Svårigheter

Det finns flera taxonomier för defekter och de kan fokusera på speciell testning, till exempel användbarhet. Det är viktigt att välja en taxonomi som är tillämplig på programvaran under test (om någon finns tillgänglig). Till exempel kanske det inte finns några taxonomier tillgängliga för innovativ programvara. Vissa organisationer har sammanställt sina egna taxonomier av troliga eller ofta observerade defekter. Oavsett vilken taxonomi som används är det viktigt att definiera den förväntade täckningen innan testningen påbörjas.

Täckning

Tekniken ger täckningskriterier som används för att bestämma när alla användbara testfall har identifierats. Täckningsobjekt kan vara strukturella element, specifikationselement, användarscenarier eller någon kombination av dessa, beroende på defektlistan. I praktiken tenderar täckningskriterierna för defektbaserade testtekniker att vara mindre systematiska än för black-box-testtekniker genom att endast allmänna regler för täckning ges och det specifika beslutet om vad som utgör gränsen för användbar täckning är godtyckligt. Liksom med andra tekniker betyder täckningskriterierna inte att

hela uppsättningen tester är komplett, utan snarare att defekter inte längre ger bidrag till användbara tester baserade på den tekniken.

Typ av defekter

Vilka typer av defekter som upptäcks beror vanligtvis på den defekttaxonomi som används. Om till exempel en lista med användargränssnittsdefekter används skulle majoriteten av de upptäckta defekterna sannolikt vara användargränssnittsrelaterade, men andra defekter kan upptäckas som en biprodukt av den specifika testningen.

3.4 Tillämpning av den mest lämpliga tekniken

Black-box och erfarenhetsbaserade testtekniker är mest effektiva när de används tillsammans. Erfarenhetsbaserade testtekniker fyller luckorna i täckningen från någon möjlig systematisk svaghet i black-box-testtekniker.

Det finns inte en perfekt teknik för alla situationer. Det är viktigt för testanalytikern att förstå fördelarna och nackdelarna med varje teknik och att kunna välja den bästa tekniken eller uppsättning av tekniker för situationen, med hänsyn till projekttyp, tidsplan, tillgång till information, testarnas skicklighet och andra faktorer som kan påverka urvalet.

I diskussionen om varje black-box och erfarenhetsbaserad testteknik (se Kapitel 3.2 respektive 3.3), hjälper informationen under "tillämpning", "begränsningar/svårigheter" och "täckning" testanalytikern att välja de lämpligaste testtekniker.

4. Testning av programvarans kvalitetsegenskaper - 180 minuter.

Nyckelord

användargränssnittets attraktivitet, användarupplevelse, användbarhet, driftduqlighet, funktionell kompletthet, funktionell korrekthet, funktionell lämplighet, funktionell ändamålsenlighet, inlärningsmöjlighet, interoperabilitet, kompatibilitet, skydd mot felanvändning, Software Usability Measurement Inventory (SUMI), tillgänglighet, Website Analysis and Measurement Inventory (WAMMI)

Utbildningsmål för testning av programvarans kvalitetsegenskaper

4.1 Introduktion

Inga utbildningsmål

4.2 Kvalitetsegenskaper för testning av affärsdomäner

- TA-4.2.1 (K2) Förklara vilka testtekniker som är lämpliga för att testa den funktionella fullständigheten, den funktionella korrektheten och den funktionella ändamålsenligheten
- TA-4.2.2 (K2) Definiera de typiska defekter som ska fokuseras på vid test av egenskaperna för funktionell fullständighet, funktionell korrekthet och funktionell ändamålsenlighet
- TA-4.2.3 (K2) Definiera när egenskaperna för funktionell fullständighet, korrekthet och ändamålsenlighet ska testas i programvarans utvecklingslivscykel
- TA-4.2.4 (K2) Förklara de angreppssätt som skulle vara lämpliga för att verifiera och validera både för implementeringen av användbarhetskraven och för uppfyllandet av användarens förväntningar
- TA-4.2.5 (K2) Förklara testanalytikerns roll i interoperabilitetstestning, inklusive identifiering av defekter inom området
- TA-4.2.6 (K2) Förklara testanalytikerns roll i portabilitetstestning inklusive identifiering av defekter inom området
- TA-4.2.7 (K4) För en given uppsättning av krav, bestämma testvillkoren som krävs för att verifiera funktionella och/eller icke-funktionella kvalitetsegenskaper inom testanalytikerns område

4.1 Introduktion

Medan det föregående kapitlet beskrev specifika tekniker användbara för testaren, berör detta kapitel tillämpningen av dessa tekniker för att utvärdera egenskaperna som används för att beskriva kvaliteten hos programvaruapplikationer eller system.

Denna kursplan berör de kvalitetsegenskaper som kan utvärderas av en testanalytiker. Attributen som ska utvärderas av den tekniska testanalytikern behandlas i Syllabus för Advanced Technical Test Analyst [CTAL-TTA].

Beskrivningen av produktkvalitetsegenskaperna i ISO 25010 [ISO25010] används som en guide för att beskriva egenskaperna. ISO-programvarukvalitetsmodellen delar upp produktkvalitet i olika produktkvalitetsegenskaper, som var och en kan ha underegenskaper. Dessa visas i tabellen nedan, tillsammans med en indikation på vilka egenskaper/underegenskaper som täcks av Syllabus/Kursplan för Test Analyst och Technical Test Analyst:

Egenskap	Underegenskaper	Test Analyst	Technical Test Analyst
Funktionell lämplighet	Funktionell korrekthet, funktionell ändamålsenlighet, funktionell kompletthet	X	
Tillförlitlighet	Mognad, feltolerans, återhämtningsförmåga, tillgång		X
Användbarhet	Igenkänning av lämplighet, inlärningsmöjlighet, driftsduglighet, användargränssnittets attraktivitet, skydd mot felanvändning, tillgänglighet	X	
Prestandaeffektivitet	Tidsbeteende, resursanvändning, kapacitet		X
Underhållbarhet	Analyserbarhet, förändringsbarhet, testbarhet, modularitet, återanvändbarhet		X
Portabilitet	Anpassningsbarhet, installationsbarhet, ersättningsbarhet	X	X
Informationssäkerhet	Konfidentialitet, integritet, icke-förnekande, ansvarighet, autenticitet		X
Kompatibilitet	Samexisterande		X
	Interoperabilitet	X	

Även om denna arbetsfördelning kan variera i olika organisationer, är det den som följs i de tillhörande ISTQB®-kursplanerna.

För alla kvalitetsegenskaper och underegenskaper som diskuteras i detta avsnitt måste de typiska riskerna vara kända så att en lämplig teststrategi kan utformas och dokumenteras. Kvalitetsegenskapstestning kräver särskild uppmärksamhet på SDLC-timing, nödvändiga verktyg, tillgång till programvara och dokumentation och teknisk expertis. Utan en strategi för att hantera varje egenskap och dess unika testbehov kanske testaren inte har tillräcklig planering, förberedelse och testexekveringstid med i tidsplanen [Bath14]. En del av denna testning, till exempel användbarhetstestning, kan kräva tilldelning av speciella personalresurser, omfattande planering, dedikerade testmiljöer, specifika verktyg, specialiserade testfärdigheter och i de flesta fall mycket tid. I vissa fall kan användbarhetstest exekveras av en separat grupp användbarhets- eller användarupplevelseexperter.

Även om testanalytikern kanske inte ansvarar för de kvalitetsegenskaper som kräver en mer teknisk strategi, är det viktigt att testanalytikern är medveten om de andra egenskaperna och förstår de överlappande områdena för testning. Till exempel kan ett testobjekt som misslyckas i

prestandatestningen sannolikt också misslyckas i användbarhetstestningen om det är för långsamt för användaren att använda effektivt. På liknande sätt är ett testobjekt med interoperabilitetsproblem med vissa komponenter förmodligen inte redo för portabilitetstestning eftersom det tenderar att dölja de mer grundläggande problemen när miljön ändras.

4.2 Kvalitetssegenskaper för testning av affärsdomäner

Testning av funktionell lämplighet är ett primärt fokus för testanalytikern. Den är inriktad på "vad" testobjektet gör. Testbasen för testning av funktionell lämplighet kan generellt vara krav, en specifikation, specifik domänexpertis eller underförstådda behov. Tester för funktionell lämplighet varierar beroende på testnivån där de utförs och kan också påverkas av SDLC. Till exempel kommer test av funktionell lämplighet som utförs under integrationstestning att testa funktionaliteten hos samverkande komponenter som implementerar en enda definierad funktion. På systemtestnivån avser det test av systemets funktionella lämplighet som helhet. För system av system kommer testning av funktionell lämplighet främst att fokusera på end-to-end-testning över de integrerade systemen. En mängd olika testtekniker används vid testning av funktionell lämplighet (se Kapitel 3).

I agil programvaruutveckling innefattar testning av funktionell lämplighet vanligtvis följande:

- Testa den specifika funktionaliteten (till exempel användarberättelser) planerad för implementering i den specifika iterationen
- Regressionstestning av all oförändrad funktionalitet

Förutom funktionell lämplighetstestning som behandlas i detta avsnitt, finns det också vissa kvalitetssegenskaper som ingår i testanalytikerns ansvarsområde som anses vara icke-funktionella (fokuserade på "hur" testobjektet levererar funktionaliteten) testområden.

4.2.1 Testning av funktionell korrekthet

Funktionell korrekthet innebär att verifiera applikationens efterlevnad av de angivna eller underförstådda kraven och kan också inkludera beräkningsnoggrannhet. Testning av funktionell korrekthet använder många av de testtekniker som förklaras i Kapitel 3 och använder ofta specifikationen eller ett äldre system som testorakel. Testning av funktionell korrekthet kan utföras på vilken testnivå som helst och är inriktad på felaktig hantering av data eller situationer.

4.2.2 Testning av funktionell ändamålsenlighet

Testning av funktionell ändamålsenlighet innebär att utvärdera och validera ändamålsenligheten för en uppsättning av funktioner för dess avsedda specificerade uppgifter. Denna testning kan baseras på den funktionella designen (till exempel användningsfall och/eller användarberättelser). Testning av funktionell ändamålsenlighet utförs vanligtvis under systemtestning, men kan också utföras under de senare stadierna av integrationstestning. Defekter som upptäcks i denna testning är indikationer på att systemet inte kommer kunna uppnå användarens behov på ett sätt som kommer att anses acceptabelt.

4.2.3 Testning av funktionell kompletthet

Testning av funktionell kompletthet görs för att bestämma täckningen av specificerade uppgifter och användarmål med den implementerade funktionaliteten. Spårbarhet mellan specifikationsobjekt (till exempel krav, användarberättelser, användningsfall) och den implementerade funktionaliteten (till exempel funktion, komponent, arbetsflöde) är avgörande för att uppnå önskad funktionell kompletthet. Mätning av funktionell kompletthet kan variera beroende på den specifika testnivån och/eller den SDLC som används. Till exempel kan funktionell kompletthet för agil programvaruutveckling baseras på implementerade användarberättelser och features. Funktionell kompletthet för systemintegrationstestning kan fokusera på täckningen av högnivåprocesser för affärsnyttan.

Att bestämma funktionell kompletthet stöds vanligtvis av testhanteringsverktyg om testanalytikern upprätthåller spårbarheten mellan testfallen och de funktionella specifikationselementen. Lägre än förväntade nivåer av funktionell kompletthet är indikationer på att systemet inte har implementerats fullt ut.

4.2.4 Interoperabilitetstestning

Interoperabilitetstestning verifierar utbytet av information mellan två eller flera system eller komponenter. Testerna fokuserar på förmågan att utbyta information och sedan använda den information som har utbytts. Testning bör täcka alla avsedda målmiljöer (inklusive variationer i hårdvara, programvara, middleware, operativsystem etc.) för att säkerställa att datautbytet fungerar korrekt. I verkligheten kan detta bara vara möjligt för ett relativt litet antal miljöer. I så fall kan interoperabilitetstestning begränsas till en utvald representativ grupp av miljöer. Att specificera tester för interoperabilitet kräver att kombinationer av de avsedda målmiljöerna identifieras, konfigureras och är tillgängliga för testteamet. Dessa miljöer testas sedan med hjälp av ett urval av testfall för funktionell lämplighet som använder de olika datautbytespunkter som finns i miljön.

Interoperabilitet handlar om hur olika komponenter och programvarusystem interagerar med varandra. Programvara med goda interoperabilitetsegenskaper kan integreras med ett antal andra system utan att kräva större förändringar eller signifikant påverkan på icke-funktionellt beteende. Antalet förändringar och arbetsinsats som krävs för att implementera och testa dessa förändringar kan användas som ett mått på interoperabilitet.

Testning av interoperabilitet för programvara kan till exempel fokusera på följande designfeatures:

- Användning av branschövergripande kommunikationsstandarder, till exempel XML
- Förmågan att automatiskt upptäcka kommunikationsbehoven för de system det interagerar med och justera därefter

Interoperabilitetstestning kan vara särskilt betydelsefull för följande:

- Programvaruprodukter och verktyg som är kommersiellt från hyllan
- Applikationer baserade på ett system av system
- System baserade på Internet of Things
- Webbtjänster med anslutning till andra system

Denna typ av testning exekveras under komponentintegrationstestning och systemintegrationstestning. På systemintegrationsnivån görs den för att bestämma hur väl det fullt utvecklade systemet interagerar med andra system. Eftersom system kan samverka på flera nivåer måste testanalytikern förstå dessa interaktioner och kunna skapa villkor för att genomföra de olika interaktionerna. Om till exempel två system kommer att utbyta data måste testanalytikern kunna skapa nödvändiga data och de transaktioner som krävs för att utföra datautbytet. Det är viktigt att komma ihåg att alla interaktioner kanske inte tydligt specificeras i kravdokumenten. Istället definieras många av dessa interaktioner endast i systemarkitekturen och designdokumentet. Testanalytikern måste kunna och vara beredd på att undersöka dessa dokument för att bestämma punkterna för informationsutbyte mellan system och mellan systemet och dess miljö för att säkerställa att alla testas. Tekniker som ekvivalensklassindelning, gränsvärdesanalys, beslutstabeller, tillståndsövergångsdiagram, användningsfall och parvis testning är alla tillämpningsbara för interoperabilitetstestning. Typiska defekter som hittas inkluderar inkorrekt datautbyte mellan interagerande komponenter.

4.2.5 Användbarhetsutvärdering

Testanalytiker har ofta rollen att samordna och stödja utvärderingen av användbarhet. Detta kan inkludera att specificera användbarhetstester eller fungera som en moderator som arbetar med användarna för att genomföra tester. För att göra detta effektivt måste testanalytikern förstå de huvudsakliga aspekterna, målen och metoderna som är involverade i dessa typer av testning. Se ISTQB® Specialistkursplanen för användbarhetstestning [ISTQB_UT_SYL] för mer information utöver beskrivningen i detta avsnitt.

Det är viktigt att förstå varför användare kan ha svårt att använda systemet eller inte har en positiv användarupplevelse (UX), till exempel med att använda programvara för underhållning. För att få denna förståelse är det först nödvändigt att inse att uttrycket "användare" kan gälla för ett brett spektrum av olika typer av personligheter, allt från IT-expert till barn till personer med funktionshinder.

4.2.5.1 Användbarhetsaspekter

Följande är de tre aspekterna som behandlas i detta avsnitt:

- Användbarhet
- Användarupplevelse (UX)
- Tillgänglighet

Användbarhet

Användbarhetstestning inriktar sig på programvarudefekter som påverkar användarens förmåga att utföra sina uppgifter via användargränssnittet. Sådana defekter kan påverka användarens förmåga att uppnå sina mål effektivt eller till belåtenhet. Användbarhetsproblem kan leda till förvirring, fel, försening eller direkt misslyckande med att utföra en del av användarens uppgifter.

Följande är de individuella underregenskaperna av användbarhet [ISO 25010]; för deras definitioner se [ISTQB GLOSSARY]

- Igenkänning av lämplighet (dvs. begriplighet)
- Lärbarhet
- Driftsduglighet
- Användargränssnittets estetik (dvs. attraktivitet)
- Skydd mot felanvändning
- Tillgänglighet (se nedan)

Användarupplevelse (UX)

Utvärdering av användarupplevelse adresserar hela användarupplevelsen med testobjektet, inte bara den direkta interaktionen. Detta är särskilt viktigt för testobjekt där faktorer som belåtenhet och användartillfredsställelse är avgörande för affärsframgång.

Typiska faktorer som påverkar användarupplevelse inkluderar följande:

- Varumärkesbild (dvs. användarens förtroende för tillverkaren)
- Interaktivt beteende
- Testobjektets hjälpsamhet, inklusive hjälpsystem, support och utbildning

Tillgänglighet

Det är viktigt att överväga tillgängligheten till programvara för dem med särskilda behov eller begränsningar för dess användning. Detta inkluderar dem som har funktionshinder.

Tillgänglighetstestning bör ta hänsyn till relevanta standarder, till exempel Web Content Accessibility Guidelines (WCAG) och lagstiftning, såsom Disability Discrimination Act (Nordirland, Australien), Equality Act 2010 (England, Skottland, Wales) och Section 508 (USA). Tillgänglighet, liksom användbarhet, måste tas i beaktande när man utför designaktiviteter. Testning sker ofta på integrationsnivåerna och fortsätter genom systemtestning och in i acceptanstestningsnivån. Defekter

noteras vanligtvis när programvaran inte uppfyller de angivna reglerna eller standarderna som definierats.

Typiska åtgärder för att förbättra tillgängligheten fokuserar på möjligheterna att ge användare med funktionshinder möjlighet att interagera med applikationen. Dessa inkluderar följande:

- Röstigenkänning av indata
- Se till att innehåll som presenteras utan text för användaren har ett likvärdigt textalternativ
- Möjliggöra att ändra textstorlek utan att förlora innehåll eller funktionalitet

Riktlinjer för tillgänglighet stöder testanalytikern genom att tillhandahålla en informationskälla och checklistor som kan användas för testning (exempel på riktlinjer för tillgänglighet finns i [ISTQB_UT SYL]). Dessutom finns verktyg och webbläsartillägg tillgängliga för att hjälpa testare att identifiera tillgänglighetsproblem, till exempel inte optimalt färgval på webbsidor som bryter mot riktlinjerna för färgblindhet.

4.2.5.2 Metoder för användbarhetsutvärdering

Användbarhet, användarupplevelse och tillgänglighet kan testas med hjälp av en eller flera av följande metoder:

- Användbarhetstestning
- Användbarhetsgranskning
- Användbarhetsundersökningar och frågeformulär

Användbarhetstestning

Användbarhetstestning utvärderar hur användarna kan använda eller lära sig att använda systemet för att nå ett specifikt mål i ett specifikt sammanhang. Användbarhetstestning inriktar sig på att mäta följande:

- Måloppfyllelse (effectiveness) - testobjektets förmåga att göra det möjligt för användare att uppnå specificerade mål med noggrannhet och fullständighet i ett specifikt användningsområde
- Verkningsgrad (efficiency) - testobjektets förmåga att göra det möjligt för användare att spendera lämpliga mängder resurser i förhållande till effektiviteten som uppnås i ett specificerat användningsområde
- Tillfredsställelse - testobjektets förmåga att tillfredsställa användare i ett specifikt användningsområde

Det är viktigt att notera att designa och specificera användbarhetstester ofta utförs av testanalytikern i samarbete med testare som har särskilda kunskaper i användbarhetstestning och användbarhetsdesigningenjörer som förstår den användarcentrerade designprocessen (se [ISTQB_UT SYL] för mer information).

Användbarhetsgranskning

Inspektioner och granskningar är en typ av testning som utförs ur ett användbarhetsperspektiv som hjälper till att öka användarens engagemangsnivå. Detta kan vara kostnadseffektivt genom att hitta användbarhetsproblem i kravspecifikationer och design tidigt i SDLC. Heuristisk utvärdering (systematisk inspektion av en användargränssnittsdesign för användbarhet) kan användas för att hitta användbarhetsproblemen i designen så att de kan adresseras som en del av en iterativ designprocess. Detta innebär att en liten antal utvärderare granskar gränssnittet och bedömer dess överensstämmelse med erkända användbarhetsprinciper ("heuristiker"). Granskningar är mer effektiva när användargränssnittet är mer synligt. Exempelvis är skärmbildsexempel vanligtvis lättare att förstå och tolka än att bara beskriva funktionaliteten som ges av en viss skärm. Visualisering är viktig för en adekvat användbarhetsgranskning av dokumentationen.

Användbarhetsundersökningar och frågeformulär

Undersökning- och frågeformulärstekniker kan tillämpas för att samla observationer och feedback om användares beteende med systemet. Standardiserade och offentligt tillgängliga undersökningar som SUMI (Software Usability Measurement Inventory) och WAMMI (Website Analysis and Measurement Inventory) möjliggör benchmarking mot en databas med tidigare användbarhetsmätningar. Eftersom SUMI tillhandahåller konkreta mätningar av användbarhet kan detta dessutom ge en uppsättning av avsluts-/acceptanskriterier.

4.2.6 Portabilitetstestning

Portabilitetstester avser till vilken grad en programvarukomponent eller system kan överföras till dess avsedda miljö, antingen som en ny installation, eller från en befintlig miljö.

ISO 25010-klassificeringen av produktkvalitetsegenskaper inkluderar följande underenskaper för portabilitet:

- Installationsbarhet
- Anpassningsbarhet
- Ersättningsbarhet

Uppgiften att identifiera risker och designa tester för portabilitetsegenskaper delas mellan testanalytikern och den tekniska testanalytikern (se [ISTQB_ALTTA_SYL] Kapitel 4.7).

4.2.6.1 Installationsbarhetstestning

Installationsbarhetstestning utförs på programvaran och skriftliga procedurer som används för att installera och avinstallera programvaran i dess målmiljö.

De typiska testmålen som är i fokus för testanalytikern inkluderar:

- Validera att olika konfigurationer av programvaran kan installeras framgångsrikt. Där ett stort antal parametrar kan konfigureras, kan testanalytikern designa tester med pairwise-tekniken för att minska antalet testade parameterkombinationer och fokusera på speciella konfigurationer av intresse (till exempel de som ofta används).
- Testa den funktionella korrektheten för installations- och avinstallationsprocedurer.
- Exekvera tester för funktionell lämplighet efter en installation eller avinstallation för att upptäcka eventuella defekter som kan ha uppkommit (till exempel felaktiga konfigurationer, icke-tillgängliga funktioner).
- Identifiera användbarhetsproblem i installations- och avinstallationsprocedurer (till exempel för att bekräfta att användare har fått förstäligen instruktioner och feedback/felmeddelanden när proceduren exekveras).

4.2.6.2 Anpassningsbarhetstestning

Anpassningsbarhetstestning kontrollerar om en given applikation kan tillämpas effektivt och ändamålsenligt för att fungera korrekt i alla avsedda målmiljöer (hårdvara, programvara, middleware, operativsystem, moln, etc.). Testanalytikern stöder anpassningsbarhetstestning genom att identifiera målmiljöerna (till exempel versioner av olika mobila operativsystem som stöds, olika versioner av webbläsare som kan användas), och designa tester som täcker dessa miljöers kombinationer. Målmiljöerna testas sedan med hjälp av ett urval av testfall för funktionella lämplighet som exekverar de olika komponenterna som finns i miljön.

4.2.6.3 Ersättningsbarhetstestning

Ersättningsbarhetstestning fokuserar förmågan hos programvarukomponenter eller versioner i ett system att bytas ut mot andra. Detta kan vara särskilt relevant för systemarkitekturer baserade på Internet of Things, där utbyte av olika hårdvaruenheter och/eller programvaruinstallationer förekommer ofta. Till exempel kan en hårdvaruenhet som ska användas i ett lager för att registrera och kontrollera lagernivåer ersättas av en mer avancerad hårdvaruenhet (till exempel med en bättre skanner) eller kan den installerade programvaran uppgraderas med en ny version som möjliggör lagerutbytesorder att automatiskt utfärdas till ett leverantörssystem.

Testning av ersättningsbarhet kan exekveras av testanalytikern parallellt med funktionell integreringstestning där mer än en alternativ komponent är tillgänglig för integration i det kompletta systemet.

5. Granskningar - 120 minuter.

Nyckelord

Checklistebaserad granskning

Utbildningsmål för granskningar

5.1 Introduktion

Inga utbildningsmål

5.2 Använda checklistor i granskningar

TA-5.2.1 (K3) Identifiera problem i en kravspecifikation enligt checklisteinformation i kursplanen

TA-5.2.2 (K3) Identifiera problem i en användarberättelse enligt checklisteinformation i kursplanen

5.1 Introduktion

Testanalytikern måste vara aktiv deltagare i granskningsprocessen och ge sina unika synpunkter. Om granskningarna görs ordentligt kan man vara den enskilt största och mest kostnadseffektiva bidragsgivaren till den totalt levererade kvaliteten.

5.2 Använda checklistor i granskning

Checklistebaserad granskning är den vanligaste tekniken som används av en testanalytiker när granskning av testbasen sker. Checklistor används under granskningar för att påminna deltagarna om att kontrollera specifika punkter under granskningen. De kan också hjälpa till att göra granskningen mindre personlig (till exempel "Det här är samma checklista som vi använder för varje granskning. Vi fokuserar inte enbart på din arbetsprodukt.").

Checklistebaserad granskning kan genomföras på ett generellt sätt för alla granskningar eller kan fokusera på specifika kvalitetsegenskaper, områden eller dokumenttyper. Till exempel kan en generisk checklista verifiera de allmänna dokumentegenskaperna, som att ha en unik identifierare, inga referenser märkta som "ska beslutas", rätt formatering och liknande. En specifik checklista för ett kravdokument kan innehålla punkter för korrekt användning av termerna "ska" och "bör", kontroll av testbarheten för varje angivet krav och så vidare.

Kravformatet kan också indikera vilken typ av checklista som ska användas. Ett kravdokument som är i berättande textformat kommer ha andra granskningskriterier än ett som är baserat på diagram.

Checklistor kan också vara inriktade på en viss aspekt, till exempel:

- En programmerares/arkitekts kompetens eller en testares kompetens – i fallet med en testanalytiker bör testarens kompetens vara den mest lämpliga
- En viss risknivå (till exempel i säkerhetskritiska system) - checklistorna innehåller vanligtvis den specifika informationen som behövs för risknivån
- En specifik testteknik - checklistan kommer att fokusera på den information som behövs för en viss teknik (till exempel regler som ska representeras i en beslutstabell)
- Ett särskilt specifikationsobjekt, till exempel krav, användningsfall eller användarberättelse - dessa berörs i följande kapitel och har i allmänhet ett annat fokus än de som används av en teknisk testanalytiker för granskning av kod eller arkitektur

5.2.1 Kravgranskningar

Följande objekt är ett exempel på vad en kravorienterad checklista kan inkludera:

- Källa till kravet (till exempel person, avdelning)
- Testbarhet för varje krav
- Prioritering för varje krav
- Acceptanskriterier för varje krav
- Tillgång till en struktur för anrop av användningsfall, om tillämpningsbart
- Unik identifiering av varje krav/användningsfall/användarberättelse
- Versionshantering av varje krav/användningsfall/användarberättelse
- Spårbarhet för varje krav från affärs-/marknadsföringskrav
- Spårbarhet mellan krav och/eller användningsfall (om tillämpningsbart)
- Användning av konsekvent terminologi (till exempel använder en ordlista)

Det är viktigt att komma ihåg att om ett krav inte är testbart, vilket innebär att det definieras på ett sådant sätt att testanalytikern inte kan avgöra hur det ska testas, så finns det en defekt i detta krav. Exempelvis är ett krav som säger "Programvaran ska vara mycket användarvänlig" inte testbart. Hur kan testanalytikern avgöra om programvaran är användarvänlig eller till och med väldigt

användarvänlig? Om kravet istället säger "Programvaran måste överensstämma med användbarhetsstandarderna som anges i dokumentet för användbarhetsstandard, version xxx", och om användbarhetsstandarddokumentet verkligen finns, är detta ett testbart krav. Det är också ett övergripande krav eftersom detta krav gäller för varje objekt i gränssnittet. I detta fall skulle detta krav enkelt kunna få många individuella testfall i en icke-trivial applikation. Spårbarhet från detta krav, eller kanske från användbarhetsstandarddokumentet, till testfallen är också kritiskt eftersom om den refererade användbarhetsspecifikationen skulle ändras, måste alla testfall granskas och uppdateras vid behov.

Ett krav är inte heller testbart om testaren inte kan avgöra om testfallet har godkänts eller underkänts eller inte kan konstruera ett test som kan godkännas eller underkännas. Till exempel "System ska vara tillgängligt 100% av tiden, 24 timmar per dag, 7 dagar i veckan, 365 (eller 366) dagar om året" är inte testbart.

En enkel checklista¹ för granskningar av användningsfall kan innehålla följande frågor:

- Är huvudbeteendet (väg) tydligt definierat?
- Är alla alternativa beteenden (vägar) identifierade, kompletta med felhantering?
- Är användargränssnittsmeddelanden definierade?
- Finns det bara ett huvudbeteende (det borde finnas, annars finns det flera användningsfall)?
- Är varje beteende testbart?

5.2.2 Granskningar av användarberättelser

I agil programvaruutveckling är kraven vanligtvis i formen av användarberättelser. Dessa berättelser representerar små enheter av bevisbar funktionalitet. Medan ett användningsfall är en användartransaktion som korsar flera funktionsområden, är en användarberättelse en mer isolerad feature och inriktningen är generellt fastställd under tiden det tog att utveckla den. En checklista² för en användarberättelse kan innehålla följande:

- Är berättelsen lämplig för mål-iterationen/sprinten?
- Är berättelsen skriven från synvinkeln av den person som begärde den?
- Är acceptanskriterierna definierade och testbara?
- Är featuren tydligt definierad och distinkt?
- Är berättelsen oberoende av andra?
- Är berättelsen prioriterad?
- Följer berättelsen det vanligt använda formatet:
Som en < användartyp >, vill jag < något mål > så att < någon anledning > [Cohn04]

Om berättelsen definierar ett nytt gränssnitt skulle det vara lämpligt att använda en generisk berättelsechecklista (som den ovan) och en detaljerad checklista för användargränssnittet.

5.2.3 Att skräddarsy checklistor

En checklista kan skräddarsys utifrån följande:

- Organisation (till exempel med tanke på företagets policy, standarder, konventioner, juridiska begränsningar)
- Projekt/utvecklingsinsats (till exempel fokus, tekniska standarder, risker)
- Den arbetsprodukt som granskas (till exempel kan kodgranskningar anpassas till specifika programmeringsspråk)
- Risknivån för arbetsprodukten som granskas

¹ Examensfrågan kommer att tillhandahålla en delmängd av checklistan för användningsfall med vilket frågan besvaras

- Testtekniker som ska användas

Bra checklistor kommer att hitta problem och också hjälpa till att starta diskussioner om andra objekt som kanske inte har nämnts specifikt i checklistan. Att använda en kombination av checklistor är ett bra sätt att se till att en granskning uppnår den högsta möjliga kvaliteten hos arbetsprodukten. Att använda checklistebaserad granskning med standardchecklistor som de som nämns i Foundation Level-kursplanen och att utveckla organisationsspecifika checklistor, som de som visas ovan, hjälper testanalytikern att vara effektiv i granskningarna.

För mer information om granskningar och inspektioner se [Gilb93] och [Wiegers03]. Ytterligare exempel på checklistor kan erhållas från referenserna i Kapitel 7.4.

6. Testverktyg och automatisering - 90 minuter.

Nyckelord

nyckelordsdriven testning, testdataförberedelse, testdesign, testexekvering, testskript

Utbildningsmål för testverktyg och automatisering

6.1 Introduktion

Inga utbildningsmål

6.2 Nyckelordsdriven automatisering

TA-6.2.1 (K3) För ett givet scenario bestämma de lämpliga aktiviteterna för en testanalytiker i ett nyckelordsdrivet testprojekt

6.3 Typ av testverktyg

TA-6.3.1 (K2) Förklara användningen och typerna av testverktyg som används i testdesign, testdataförberedelse och testexekvering

6.1 Introduktion

Testverktyg kan markant förbättra testningens effektivitet och noggrannhet. Testverktygen och automatiseringsmetoderna som används av en testanalytiker beskrivs i detta kapitel. Det bör nämnas att testanalytikern arbetar tillsammans med utvecklare, testautomatiserare och tekniska testanalytiker för att skapa testautomatiseringsslösningar. Nyckelordsdriven automatisering i synnerhet involverar testanalytikern och utnyttjar hans erfarenhet inom verksamheten och systemfunktionaliteten.

Mer information om ämnet av testautomatisering och om rollen som testautomatiserare finns i ISTQB® Advanced Level Test Automation Engineer-kursplanen [ISTQB_ALTAE_SYL].

6.2 Nyckelordsdriven Testning

Nyckelordsdriven testning är en av de viktigaste testautomatiseringssmetoderna och involverar testanalytikern i att tillhandahålla huvudindata: nyckelord och data.

Nyckelord (ibland kallade action word) används oftast, men inte enbart, för att representera affärsinteraktioner på hög nivå med ett system (till exempel "cancel order"). Varje nyckelord används vanligtvis för att representera ett antal detaljerade interaktioner mellan en aktör och systemet som testas. Sekvenser av nyckelord (inklusive relevant testdata) används för att specificera testfall [Buwalda02].

I testautomatisering implementeras ett nyckelord som ett eller flera exekverbara testskript. Verktyg läser testfall skrivna som en sekvens av nyckelord som kallar på lämpliga testskript som implementerar nyckelordsfunktionaliteten. Skripten implementeras på ett mycket modulärt sätt för att möjliggöra enkel kartläggning till specifika nyckelord. Programmeringsfärdigheter krävs för att implementera dessa modulära skript.

Följande är de främsta fördelarna med nyckelordsdriven testning:

- Nyckelord som relaterar till en viss applikation eller affärsdomän kan definieras av domänexperter. Detta kan göra arbetet med testfallsspecifikationen mer effektiv.
- En person med främst domänexpertis kan dra nytta av automatisk testfallsexekvering (när nyckelorden har implementerats som skript) utan att behöva förstå den underliggande automatiseringskoden
- Att använda en modulär beskrivningsteknik möjliggör effektivt underhåll av testfall för testautomatiseraren när förändringar i funktionaliteten och gränssnittet till programvaran som testas inträffar [Bath14].
- Testfallsspecifikationer är självständiga från deras implementering.

Testanalytiker skapar och underhåller vanligtvis data om nyckelord. De måste inse att arbetet med skriptutveckling fortfarande är nödvändigt för att implementera nyckelorden. När nyckelorden och data som ska användas har definierats, översätter testautomatiseraren (eller tekniska testanalytikern) affärsprocessens nyckelord och åtgärder på lägre nivå till automatiserade testskript.

Medan nyckelordsdriven testning vanligtvis exekveras under systemtestning kan kodutveckling börja samtidigt med testdesignen. I en iterativ miljö, särskilt när kontinuerlig integration/implementering används, är testautomatiseringsutvecklingen en kontinuerlig process.

När input-nyckelorden och data har skapats tar testanalytikern ansvaret för att exekvera de nyckelordsdrivna testfallen och att analysera de potentiella felsymptom som kan uppstå.

När en anomaly upptäcks ska testanalytikern hjälpa till med att undersöka orsaken till felsymptomet för att se om felet ligger i nyckelorden, indata, själva testautomatiseringsskriptet eller med applikationen som testas. Vanligtvis är det första steget i felsökning att exekvera samma test med samma data

manuellt för att se om felet är i själva applikationen. Om detta inte ger ett felsymptom ska testanalytikern granska sekvensen av tester som ledde fram till felsymptomen för att avgöra om problemet inträffade i ett tidigare steg (kanske genom att introducera felaktiga indata), men att defekten inte uppträdde förrän senare i behandlingen. Om testanalytikern inte kan bedöma orsaken till felsymptomen ska felsökningsinformationen överlämnas till den tekniska testanalytikern eller utvecklare för ytterligare analys.

6.3 Typ av testverktyg

Mycket av en testanalytikers arbete kräver effektiv användning av verktyg. Denna effektivitet förbättras av följande:

- Att veta vilka verktyg som ska användas
- Att veta att verktyg kan öka effektiviteten av testinsatsen (till exempel genom att hjälpa till att ge bättre täckning under den planerade tiden)

6.3.1 Testdesignverktyg

Testdesignverktyg används för att hjälpa till med att skapa testfall och testdata som ska tillämpas för testning. Dessa verktyg kan arbeta utifrån specifika kravdokumentformat, modeller (till exempel UML) eller indata som tillhandahålls av testanalytikern. Testdesignverktyg är ofta designade och byggda för att fungera med särskilda format och verktyg, till exempel specifika kravhanteringsverktyg.

Testdesignverktyg kan ge testanalytikern underlag för beslut om vilka typer av tester som behövs för att uppnå önskad nivå av täckning, förtroende för systemet eller åtgärder för minskning av produkt risken. Till exempel genererar (och visar) verktyg för klassifikationsträd den uppsättning kombinationer som behövs för att nå full täckning baserat på ett valt täckningskriterium. Denna information kan sedan användas av testanalytikern för att bestämma de testfall som måste exekveras.

6.3.2 Verktyg för förberedelse av testdata

Verktyg för förberedelse av testdata kan ge följande fördelar:

- Analysera ett dokument, till exempel ett kravdokument eller till och med källkoden för att bestämma de data som krävs för testningen för att uppnå en viss nivå av täckning.
- Ta en datauppsättning från ett produktionssystem och anonymisera det för att ta bort någon personlig information så det fortfarande bibehåller den interna dataintegriteten. Dessa data kan sedan användas för testning utan risk för säkerhetsläckage eller missbruk av personlig information. Detta är särskilt viktigt när stora volymer realistiska data krävs och där säkerhets- och dataintegritetsrisker gäller.
- Generera syntetiska testdata från givna uppsättningar av indataparametrar (till exempel för användning i slumpbaserad testning). Vissa av dessa verktyg kommer att analysera databasstrukturen för att bestämma vilka indata som kommer att krävas från testanalytikern.

6.3.3 Automatiserade testexekveringsverktyg

Testexekveringsverktyg används av testanalytiker på alla testnivåer för att exekvera automatiserade tester och kontrollera de faktiska resultaten. Målet med att använda ett testexekveringsverktyg är vanligtvis ett eller flera av följande:

- För att minska kostnaderna (i form av arbete och/eller tid)
- För att exekvera fler tester
- För att exekvera samma test i flera miljöer
- För att göra testexekveringen mer repeterbar
- För att exekvera tester som skulle vara omöjliga att exekvera manuellt (dvs. stora valideringstester)

Dessa mål överlappar ofta de viktigaste målen att öka täckningen och samtidigt minska kostnaderna.

Avkastningen på investeringar för testexekveringsverktyg är vanligtvis högst vid automatisering av regressionstester på grund av den låga förväntade underhållsnivån och den upprepade exekveringen av testerna. Automatisering av smoke-tester kan också vara en effektiv användning av automatisering på grund av den frekventa användning av testerna, behovet av ett snabbt testresultat och, även om underhållskostnaderna kan vara högre, förmågan att ha ett automatiserat sätt att utvärdera en ny byggnad i en kontinuerlig integrationsmiljö.

Testexekveringsverktyg används vanligtvis under system- och integrationstestning. Vissa verktyg, särskilt API-testverktyg, kan också användas i komponenttestning. Att utnyttja verktygen där de är mest tillämpningsbara hjälper till att förbättra avkastningen på investeringen.

7. Referenser

7.1 Standarder

[ISO25010] ISO/IEC 25010 (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models, Chapter 4

[ISO29119-4] ISO/IEC/IEEE 29119-4 Software and Systems Engineering – Software Testing – Part 4, Test Techniques, 2015

OMG-DMN] Object Management Group: OMG®Decision Model and Notation™, Version 1.3, December 2019; url: www.omg.org/spec/DMN/, Chapter 8

[OMG-UML] Object Management Group: OMG®Unified Modeling Language®, Version 2.5.1, December 2017; url: www.omg.org/spec/UML

[RTCA DO-178C/ED-12C]: Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE ED12C, 2013., Chapter 1

7.2 ISTQB® och IREB Dokument

[IREB_CPRE] IREB Certified Professional for Requirements Engineering Foundation Level Syllabus, Version 2.2.2, 2017

[ISTQB_AL_OVIEW] ISTQB® Advanced Level Overview, Version 2.0

[ISTQB_ALTTA_SYL] ISTQB® Advanced Level Technical Test Analyst Syllabus, Version 2019

[ISTQB_FL_SYL] ISTQB® Foundation Level Syllabus, Version 2018

[ISTQB_GLOSSARY] Standard glossary of terms used in Software Testing, url: <https://glossary.istqb.org/>

[ISTQB_TAE_SYL] ISTQB® Advanced Level Test Automation Engineer Syllabus, Version 2017

[ISTQB_UT_SYL] ISTQB® Foundation Level Specialist Syllabus Usability Testing, Version 2018

7.3 Böcker och Artiklar

[Bath14] Graham Bath, Judy McKay, "The Software Test Engineer's Handbook (2nd Edition)", Rocky Nook, 2014, ISBN 978-1-933952-24-6

[Beizer95] Boris Beizer, "Black-box Testing", John Wiley & Sons, 1995, ISBN 0-471-12094-4

[Black02]: Rex Black, "Managing the Testing Process (2nd edition)", John Wiley & Sons: New York, 2002, ISBN 0-471-22398-0

- [Black07]: Rex Black, "Pragmatic software testing: Becoming an effective and efficient test professional", John Wiley and Sons, 2007, ISBN 978-0-470-12790-2
- [Black09]: Rex Black, "Advanced Software Testing, Volume 1", Rocky Nook, 2009, ISBN 978-1-933-952-19-2
- [Buwalda02]: Hans Buwalda, "Integrated Test Design and Automation: Using the Test Frame Method", Addison-Wesley Longman, 2002, ISBN 0-201-73725-6
- [Chow1978]: T.S. Chow, "Testing Software Design Modeled by Finite-State Machines", IEEE Transactions on Software Engineering vol. SE-4, issue 3, May 1978, pp. 178-187
- [Cohn04]: Mike Cohn, "User Stories Applied: For Agile Software Development", Addison-Wesley Professional, 2004, ISBN 0-321-20568-5
- [Copeland04]: Lee Copeland, "A Practitioner's Guide to Software Test Design", Artech House, 2004, ISBN 1-58053-791-X
- [Craig02]: Rick David Craig, Stefan P. Jaskiel, "Systematic Software Testing", Artech House, 2002, ISBN 1-580-53508-9
- [Forgács19]: István Forgács, Attila Kovács, "Practical Test Design", BCS, 2019, ISBN 978-1-780-1747-23
- [Gilb93]: Tom Gilb, Graham Dorothy, "Software Inspection", Addison-Wesley, 1993, ISBN 0-201-63181-4
- [Koomen06]: Tim Koomen, Leo van der Aalst, Bart Broekman, Michiel Vroon "TMap NEXT, for result driven testing", UTN Publishers, 2006, ISBN 90-72194-80-2
- [Kuhn16]: D. Richard Kuhn et al, "Introduction to Combinatorial Testing, CRC Press, 2016, ISBN 978-0-429-18515-1
- [Myers11]: Glenford J. Myers, "The Art of Software Testing 3rd Edition", John Wiley & Sons, 2011, ISBN: 978-1-118-03196-4
- [Offutt16]: Jeff Offutt, Paul Ammann, Introduction to Software Testing – 2nd Edition, Cambridge University Press, 2016, ISBN 13: 9781107172012,
- [vanVeenendaal12]: Erik van Veenendaal, "Practical risk-based testing." Product Risk Management: The PRISMA Method ", UTN Publishers, 2012, ISBN 9789490986070
- [Wiegers03]: Karl Wiegers, "Software Requirements 2", Microsoft Press, 2003, ISBN 0-735-61879-8
- [Whittaker03]: James Whittaker, "How to Break Software", Addison-Wesley, 2003, ISBN 0-201-79619-8

[Whittaker09]: James Whittaker, "Exploratory software testing: tips, tricks, tours, and techniques to guide test design", Addison-Wesley, 2009, ISBN 0-321-63641-4

7.4 Andra referenser

Följande referenser hänvisar till information tillgänglig på Internet och andra ställen. Även om dessa referenser kontrollerades vid tidpunkten av publiceringen av denna Advanced Level-kursplan, kan ISTQB® inte hållas ansvarigt om referenserna inte längre är tillgängliga.

- Kapitel 3
 - Czerwonka, Jacek: www.pairwise.org
 - Sample defect taxonomy based on Boris Beizer's work: inet.uni2.dk/~vinter/bugtaxst.doc
 - Good overview of various taxonomies: testingeducation.org/a/bugtax.pdf
 - Heuristic Risk-Based Testing By James Bach
 - Exploring Exploratory Testing, Cem Kaner and Andy Tinkham, www.kaner.com/pdfs/ExploringExploratoryTesting.pdf
 - Pettichord, Bret, "An Exploratory Testing Workshop Report", www.testingcraft.com/exploratorypettichord
- Kapitel 5
 - <http://www.tmap.net/checklists-and-templates>

8. Appendix A

Följande tabell är härledd från den kompletta tabellen som tillhandahålls i ISO 25010. Den fokuserar endast på kvalitetsegenskaperna som täcks i Test Analyst-kursplanen och jämför termerna som används i ISO 9126 (som användes i 2012-versionen av kursplanen) med de i den nyare ISO 25010 (som användes i denna version).

ISO/IEC 25010	ISO/IEC 9126-1	Notes
Functional suitability	Functionality	
Functional completeness		
Functional correctness	Accuracy	
Functional appropriateness	Suitability	
	Interoperability	Moved to Compatibility
Usability		
Appropriateness recognizability	Understandability	New name
Learnability	Learnability	
Operability	Operability	
User error protection		New subcharacteristic
User interface aesthetics	Attractiveness	New name
Accessibility		New subcharacteristic
Compatibility		New definition
Interoperability		
Co-Existence		Covered in Technical Test Analyst

9. Index

0-switch, 29
agil utveckling, 56
anonymisera, 53
anpassningsbarhetstestning, 45
användarberättelse, 49
användarupplevelse, 43
användbarhetsutvärdering, 43
användningsfallsbaserad testning, 32
avslutskriterier, 12
beslutstabeller, 27
black-box-testtekniker, 24
breadth-first, 22
checklistebaserad granskning, 48
checklistebaserad testning, 35
checklistor i granskning, 48
defektbaserade testtekniker, 37
depth-first, 22
ekvivalensklassindelning, 24
erfarenhetsbaserad testning, 34
erfarenhetsbaserade testtekniker, 34
ersättningsbarhetstestning, 45
felgissning, 34
funktionell ändamålsenlighet, 41
funktionell kompletthet, 41
funktionell korrekthet, 41
funktionell lämplighet, 40
gränsvärdesanalys, 26
installationsbarhetstestning, 45
interoperabilitetstestning, 42
ISO 25010, 40
iterativ, 11
klassifikationsträdsteknik, 30
kombination av tekniker, 33
kvalitetsegenskaper, 39
n-switchar, 30
nyckelord, 52
nyckelordsdriven automatisering, 52
oskriptad testning, 17
parvis testning, 31
portabilitetstestning, 45
produktrisk, 21
programvarans utvecklingslivscykel, 11
riskbaserad testning, 19
riskbedömning, 21
riskidentifiering, 20
riskreducering, 21
SDLC, 11
standarder, 55
SUMI, 45
testanalys, 12
testbas, 12
testcharter, 36
testdesign, 13
testdesignverktyg, 53
testexekvering, 18
testexekveringsschema, 16
testexekveringsverktyg, 53
testfall, 15
testimplementering, 16
testmiljö, 17
testning av programvarans
 kvalitetsegenskaper, 39
Testning med hjälp av beslutstabeller, 27
testorakel, 41
testskript, 14
teststrategi, 11
testsvit, 16
testtekniker, 23
testvillkor, 13
tillämpning av den mest lämpliga tekniken,
 38
tillståndsbaserad testning, 28
utforskande testning, 36
verktyg för förberedelse av testdata, 53
WAMMI, 45