Advanced Level Syllabus
Test Analyst

Version 3.1

International Software Testing Qualifications Board

Dansk udgave v1.0

Besked om ophavsrettigheder

Denne [danske] beskrivelse af rettigheder vedr. Syllabus skal fortolkes i overensstemmelse med den engelske originaltekst.

Copyright Notice

Revisionshistorie

Revisionshistorik for Engelsk version

Version	Dato	Bemærkninger
3.1.0	3rd March 2021	Minor update with section 3.2.3 rewritten and various wording improvements
2019 v3.0	19 October 2019	Major update with overall revision and scope reduction
2012 v2.0	19 October 2012	First version as a separate AL-TA Syllabus

Revisionshistorik for Dansk oversættelse af ISTQB® CTFL 2018

Version	Dato	Bemærkninger
1.0	Juni, 2021	ISTQB® CTAL Test Analyst Advanced Level v3.1 syllabus, dansk oversættelse

Indholdsfortegnelse

Anerkendelser

Dette dokument blev formelt frigivet i sin engelske udgave af ISTQB® General Assemble (23. februar 2021).

ISTQB® CTAL-TA Test Analyst v3.1 blev produceret af et team under International Software Testing Qualification Board ved Advanced Level Working Group: Mette Bruhn-Pedersen (Working Group Chair); Matthias Hamburg (Product Owner); Wim Decoutere, István Forgács, Adam Roman, Jan Sabak og Marc-Florian Wendland (Authors)

Følgende personer deltog i review og kommentering af 3.1 versionen: Andreas Hetz, Armin Born, Attila Kovacs, Benjamin Timmermans, Blair Mo, Chen Geng (Kevin), Chenyifan, Chris van Bael, Gary Mogyorodi, Gery Ágnecz, Ingvar Nordström, Istvan Gercsák, Joan Killeen, Klaudia Dussa-Zieger, Lucjan Stapp, Marton Matyas, Meile Posthuma, Murian Song, Nishan Portoyan, Ole Chr. Hansen, Palma Polyak, Paul Weymouth, Péter Sótér, Richard Green, Rik Marselis, Stephanie van Dijck, Stuart Reid, Tal Pe'er, Tobias Horn og Zsolt Hargitai

ISTQB® CTAL-TA Test Analyst 2019 blev produceret af et team under International Software Testing Qualification Board ved Advanced Level Working Group: Graham Bath, Judy McKay og Mike Smith

Følgende personer deltog i review og kommentering af 2019 versionen: Abhishek Sharma, Adam Roman, Andrea Szabó, Attila Ko-vács, Beata Karpinska, Benjamin Timmermans, Blair Mo, Carsten Weise, Chen Geng, Chris Van Bael, Claire Lohr, David Frei, Dietrich Leimsner, Don Mills, Erik van Veenendaal, Francisca Cano Ortiz, Gary Mogyorodi, Guo Chaonian, Henriett Braunné Bokor, Ingvar Nordström, Jan Versmissen, Jan te Kock, József Kreisz, Klaudia Dussa-Zieger, Laura Albert, Lloyd Roden, Lucjan Stapp, Markus Beck, Marton Matyas, Matthias Hamburg, Meile Posthuma, Melinda Eckrich-Brajer, Milena Donato, Paul Weymouth, Pálma Polyák, Péter Földházi Jr, Péter Sótér, Ramit Manohar Kaul, Ren Liang, Rik Marselis, Robert Werkhoven, Tal Peer, Tobias Horn, Wim Decoutere, Zhai Hongbao, Zsolt Hargitai og Ágota Horváth

ISTQB® CTAL-TA Test Analyst 2019 blev produceret af et team under International Software Testing Qualification Board som en undergruppe til Advanced Level Working Group: Judy McKay (Chair), Erik van Veenendaal og Mike Smith

Da CTAL-TA 2012 blev afsluttet bestod Advanced Level Working Group af: Mike Smith (Chair), Bernard Homès (Vice Chair), Debra Friedenberg, Eric Riou du Cosquer, Erik van Veenendaal, Geoff Thompson, Graham Bath, Hans Schaefer, Jamie Mitchell, Jan Sabak, Judy McKay, Kenji Onishi, Klaus Olsen, Maria Clara Choucair, Meile Posthuma, Paul Jorgensen, Rex Black, Thomas Mueller og Tsuyoshi Yumoto

Følgende personer deltog i review og kommentering af 2012 versionen: Arne Becher, Bernard Homès, Chris van Bael, Debi Zylbermann, Don Mills, Eli Margolin, Erik van Veenendaal, Frans Dijkman, Gary Mogyorodi, Graham Bath, Hans Schaefer, Hans Weiberg, Ingvar Nordstrom, Jan Sabak, Junfei Ma, Jurian van der Laar, Kobi Halperin, Marco Sogliani, Maria Jönsson, Mats Grindal, Paul Weymouth, Piet de Roo, Raluca Madalina Popescu, Reto Mueller, Rex Black, Rik Marselis, Stefan Mohacsi, Stephanie van Dijk, Stuart Reid, Tal Pe'er, Thomas Mueller, Wenqiang Zheng og Yaron Tsubery

Oversættelse

DSTB takker Bent Hemmingsen for den danske oversættelse af ISTQB® CTAL-TA Test Analyst v2019 og Klaus Skafte for oversættelse af version v3.1.

DSTB takker følgende personer for at hjælpe med oversættelse og review af Begrebslisten i forbindelse med Certified Tester 2018 og v3.1 syllabus: Klaus Skafte (Projektleder), Ebbe Munk, Jane M. Nash, Jesper Petersen og Ole Chr. Hansen

DSTB takker følgende personer for review af den danske CTAL-TA 2019 og v3.1 oversættelse: Klaus Skafte (Projektleder), Camilla Høgild, Ebbe Munk, Jane M Nash, Lisbeth Hylke Thomsen og Ole Chr. Hansen

DSTB takker Anders Christian Boisen for den danske oversættelse af Copyright Notice.

0. Introduktion

0.1 Formålet med denne syllabus

Denne syllabus udgør grundlaget for International Software Testing Qualification for testanalytikere på Advanced Level. ISTQB® udleverer denne syllabus til følgende:

1. Lokale ISTQB®-organisationer, så det kan blive oversat til lokale sprog og godkendes af undervi-serne. De lokale organisationer kan tilpasse syllabus efter deres lokale og sproglige krav, samt til-føje henvisninger, så syllabus tilpasses til lokale publikationer
2. Eksamensudvalg, der kan udlede spørgsmål på egne lokale sprog og tilpasse disse til læringsmål for hver syllabus
3. Udbydere af undervisning, med det formål at producere kursusmaterialer og finde frem til pas-sende undervisningsmetoder
4. Certificeringskandidater, så de kan forberede sig på certificeringseksamen (som en del af et kursus eller selvstændigt)
5. Til det internationale software- og systemteknikermiljø med det formål at udvikle software- og sy-stemtest-professionen, og som grundlag for bøger og artikler

ISTQB® kan lade andre organisationer anvende syllabus til andre formål, hvis de på forhånd indhenter skrift-lig godkendelse.

0.2 Certificeret tester på Advanced Level i softwaretest

Denne syllabus på Advanced Level består af tre dele, der henvender sig til følgende roller:

- Testmanager
- Testanalytiker
- Teknisk Testanalytiker

Oversigten over ISTQB® Advanced Level er et separat dokument [ISTQB_AL_OVIEW], der indeholder føl-gende oplysninger:

- Forretningsmæssige gevinster ved hver syllabus
- Matricen viser sporbarhed mellem forretningens resultater og læringsmålsætninger
- Resumé for hver syllabus
- Forholdet mellem de forskellige pensa

0.3 Eksamensrelevante læringsmål og kognitive vidensniveauer

Læringsmålene understøtter forretningens resultater og anvendes til at fremstille eksaminer til at opnå test-analytikercertifikater på Advanced Level.

De specifikke læringsmåls vidensniveauer vises i begyndelsen af hvert kapitel og klassificeres som følgende:

- K2: Forstå
- K3: Anvende
- K4: Analysere

Definitionen på alle de begreber, der nævnes som nøgleord under kapitlernes overskrifter, skal huskes (K1), også uden at de direkte nævnes som læringsmål.

0.4 Certificeringseksamen for testanalytikere på Advanced Level

Certificeringseksamen for testanalytikere på Advanced Level er baseret på denne syllabus. Svar på de enkelte eksamensspørgsmål kan kræve anvendelse af materialer, der bygger på mere end et afsnit i denne syllabus. Alle afsnit i syllabus er eksamensrelevante, bortset fra indledningen og bilag. Standarder, bøger og andre ISTQB®-studieplaner inkluderes som henvisninger, men deres indhold er ikke eksamensrelevant ud over det, der står opsummeret i denne syllabus fra disse standarder, bøger og anden ISTQB®-pensa.

Eksamen foregår som multiple-choice. Der er 40 spørgsmål. For at bestå eksamen skal man svare korrekt på nok spørgsmål til at opnå mindst 65% af det totale antal points.

Eksamen kan tages som en del af et akkrediteret kursus eller selvstændigt (f.eks. ved et eksamenscenter eller ved en offentlig eksamen). Det er ikke noget krav at skulle have afsluttet et akkrediteret kursus inden eksamen.

0.5 Adgangskrav for eksamen

Man skal have bestået certifikat som certificeret tester på Foundation Level, inden man kan tage certificeringseksamen som testanalytiker på Advanced Level.

0.6 Forventninger til arbejdserfaring

Ingen af læringsmålene for testanalytikere på Advanced Level forudsætter at kursisten har specifikke arbejdserfaringer.

0.7 Akkreditering af kurserne

Lokale ISTQB-organisationer kan akkreditere undervisere, hvis deres undervisningsmateriale følger denne syllabus. Undervisere kan få retningslinjer fra bestyrelsen eller akkrediteringsorganet. Et akkrediteret kursus skal følge denne syllabus og får derved adgang til at afholde ISTQB® -eksaminer som en del af kurset.

0.8 Syllabus detaljeniveau

Denne syllabus detaljeniveau muliggør, at kurser og eksaminer følger en international standard. For at opnå dette mål består syllabus af:

- Overordnede instruktioner for de målsætninger, der beskriver intentionen med testanalytikercertifikatet på Advanced Level
- En liste over de begreber, som den studerende skal genkende
- Læringsmålsætninger for hvert vidensområde, der beskriver de kognitive læringsresultater, som skal opnås
- En beskrivelse af nøglekoncepter, herunder henvisninger til kilder som f.eks. den accepterede litteratur og standarder

Syllabus indhold er ikke en beskrivelse af hele vidensområdet for softwaretest; det gengiver blot det detaljeniveau, der skal dækkes i Advanced Level kurset. Det fokuserer på det materiale, der gælder for alle softwareprojekter, herunder agil softwareudvikling. Denne syllabus indeholder ikke specifikke læringsmålsætninger for nogen særlig softwareudviklings livscyklus (SDLC), men det beskrives f.eks., hvordan disse koncepter fungerer i agil softwareudvikling, andre typer af iterative og inkrementelle og sekventielle livscyklusser.

0.9 Syllabus indhold

Der er seks kapitler med eksamensrelevant indhold. Under overskriften i hvert kapitel nævnes den forventede undervisnings- og øvetid for kapitlet. For de akkrediterede kurser kræver syllabus som minimum 20 timer og 30 minutters undervisning, der fordeles over de seks kapitler på følgende måde:

- Kapitel 1: Testanalytikerens opgaver i testprocessen – 150 minutter
- Kapitel 2: Testanalytikerens opgaver i risikobaserede tests – 60minutter
- Kapitel 3: Testteknikker – 630 minutter
- Kapitel 4: Testsoftwarens kvalitetsegenskaber – 180 minutter
- Kapitel 5: Review – 120 minutter
- Kapitel 6: Testværktøjer og automatisering – 90 minutter

1. Testanalytikerens opgaver i testprocessen — 150 minutter

Nøgleord

højniveau testcase, lavniveau testcase, slutkriterier, test, testafvikling, testafviklingsplan, testanalyse, testbetingelse, testdata, testdesign, testgrundlag, testimplementering, testprocedure, testsuite

Læringsmål for testanalytikerens opgaver i løbet af testprocessen

1.1 Introduktion

Ingen læringsmål

1.2 Test i softwareudviklingens livscyklus

TA-1.2.1 (K2) Forklar hvordan og hvorfor der er forskellig timing og grad af medvirken for testanalytikeren, når denne arbejder i forskellige modeller for softwareudviklingslivscyklusser

1.3 Testanalyse

TA-1.3.1 (K2) Opsummér passende opgaver for en testanalytiker, der udfører analyse

1.4 Testdesign

TA-1.4.1 (K2) Forklar hvorfor interessenterne bør kunne forstå testbetingelser
TA-1.4.2 (K4) Vælg et passende designniveau for testcases i det pågældende projektscenarie (højniveau eller lavniveau)
TA-1.4.3 (K2) Forklar faktorer, der skal overvejes i design af testcasen

1.5 Testimplementering

TA-1.5.1 (K2) Opsummér passende opgaver for en testanalytiker, der udfører testimplementering

1.6 Testafvikling

TA-1.6.1 (K2) Opsummér passende opgaver for en testanalytiker, der afvikler test

1.1 Introduktion

I ISTQB® Foundation Level syllabus beskrives de følgende aktiviteter som en del af testprocessen:

- Testplanlægning
- Testovervågning og -kontrol
- Testanalyse
- Testdesign
- Testimplementering
- Testafvikling
- Testafslutning

I denne syllabus for testanalytikere på Advanced Level omtales de aktiviteter, der er særligt relevante for testanalytikeren, mere detaljeret. Dette giver testprocessen en højere detaljegrad, så den passer bedre til softwareudviklingscyklussens (SDLC) modeller.

At finde frem til de passende tests, designe, implementere dem, og derefter afvikle dem er de primære fokusområder for testanalytikeren. Selvom det er vigtigt at forstå de andre trin i testprocessen, drejer størstedelen af testanalytikerens arbejde sig om følgende aktiviteter:

- Testanalyse
- Testdesign
- Testimplementering
- Testafvikling

De andre aktiviteter i testprocessen er beskrevet fyldestgørende på Foundation Level og kræver ikke yderligere forklaring på dette niveau.

1.2 Test i softwareudviklingens livscyklus (SDLC)

Den overordnede SDLC bør overvejes, når teststrategien skal defineres. Det tidspunkt, hvor testanalytikeren bliver involveret, varierer for forskellige typer SDLC's; graden af involvering, arbejdstid, tilgængelige oplysninger og forventninger kan også variere en del. Testanalytikeren skal være bevidst om hvilke typer oplysninger, der skal leveres til de øvrige organisatoriske roller, som:

- Kravudarbejdelse og -styring – feedback på review af krav
- Projektledelse - planlagt bidrag
- Konfiguration og change management – resultatet af build verificeringstest, versionskontrol information
- Softwareudvikling - meddelelser om fundne defekter
- Softwarevedligeholdelse – rapporterer defekter, defektrettelses effektivitet og gentest
- Teknisk support - præcis dokumentering af workarounds og kendte problemer
- Fremstilling af dokumentation (f.eks. specifikationer for databasedesign, dokumentation af testmiljø) - input og teknisk gennemgang af disse dokumenter

Testaktiviteter skal være i overensstemmelse med den valgte SDLC, der kan være sekventiel, iterativ, inkrementel eller en hybrid af disse. F.eks. kan man i en sekventiel V-model anvende testprocessen på systemtestniveau som i det følgende:

- Systemtestplanlægningen finder løbende sted under projektplanlægningen, og testovervågning og -kontrol fortsætter, indtil testen er færdigafviklet. Det kan påvirke input til testplanen, der leveres af testanalytikeren med henblik på projektstyring
- Systemtestanalysen og -designet følger dokumenter som systemkravspecifikationer, specifikationer for system og design af systemarkitektur (højniveau), såvel som komponentdesign (lavniveau)

- Implementering af systemtestmiljø kan påbegyndes under systemdesignet, selvom hovedparten typisk finder sted samtidig med kodning og komponenttest. Arbejdet med implementering af systemtestaktiviteter strækker sig ofte indtil få dage før systemtesten begynder
- Afvikling af systemtesten begynder, når systemtestens startkriterier er opfyldt - eller hvis nogle startkriterier bortfalder. Det betyder typisk, at komponenttesten og ofte også komponentintegrationstestens slutkriterier er opfyldt. Systemtestafviklingen fortsætter, til systemtestens slutkriterier er opfyldt
- Testafslutningsaktiviteter for systemtesten finder sted når slutkriterierne er opfyldt

I nogle tilfælde følger de iterative og inkrementelle modeller ikke samme aktivitetsrækkefølge, og det kan udelukke visse aktiviteter. F.eks. kan en iterativ model anvende et reduceret sæt af testaktiviteter for hver iteration. Testanalyse, design, implementering og afvikling kan udføres i hver iteration, hvorimod planlægning på højt niveau udføres i begyndelsen af projektet, og testafslutningsopgaver udføres i slutningen.

I agil softwareudvilking er det almindeligt at anvende en mindre formaliseret proces og et tæt samarbejde med projektets interessenter. Det gør det lettere at udføre ændringer i projektet. Det kan ske, at der ikke findes en velbeskrevet rolle for testanalytikeren. Testdokumenterne er mindre detaljerede, og kommunikationen er kortere og mere regelmæssig.

Agil softwareudvikling involverer test fra begyndelsen. Testen starter samtidig med produktudviklingen, mens udviklerne udfører deres indledende arbejde på arkitektur og design: Review må ikke være formaliseret, men fortsættes, når softwaren gennemgår forandringer. Det forventes, at testanalytikeren deltager i hele projektet, og testanalyse opgaver forventes at blive udført af teamet.

Iterative og inkrementelle modeller strækker sig fra agil softwareudvikling, hvor der er en forventning om, at forandringer finder sted, mens kundens krav udvikles, til de hybridudviklingsmodeller, som f.eks. iterative/inkrementelle, kombineret med en V-model tilgang. I en sådan hybridmodel skal testanalytikerne deltage i planlægning og design aspekter i de sekventielle aktiviteter, og senere indtager testanalytikerne en mere interaktiv rolle, under de iterative/inkrementelle aktiviteter.

Uanset hvilken SDLC, der anvendes, skal testanalytikerne forstå timingen og forventningerne til dennes rolle. Testanalytikerne yder den mest effektive indsats til softwarens kvalitet ved at justere deres aktiviteter og de skal arbejde hen imod passende tidspunkter at deltage i, snarere end at være afhængige af rollen i en forudbestemt model.

1.3 Testanalyse

Testprojektets omfang defineres i løbet af testplanlægningen. Under testanalyse anvender Testanalytikerne denne definition til at:

- Analysere testgrundlaget
- Identificere forskellige defekttyper i testgrundlaget
- Identificere og prioritere testbetingelser og funktioner, der skal testes
- Dokumentere tovejs sporbarhed mellem hvert element i testgrundlaget og de tilknyttede testbetingelser
- Udføre opgaver, der er forbundet med risikobaseret test (se kapitel 2)

For at testanalytikerne kan fortsætte med testanalysen på produktiv vis, bør de følgende startkriterier være opfyldt:

- Der findes tilstrækkelig viden (f.eks. krav, userstories), om testobjektet, så det kan fungere som testgrundlag (se [ISTQB_FL_SYL] afsnit 1.4.2 og 2.2 eller en liste over andre mulige kilder med testgrundlag)
- Testgrundlaget er reviewet med fornuftige resultater og er blevet opdateret på grundlag af de behov, der har vist sig ved gennemgangen. Bemærk, at hvis man skal definere testcases på højt niveau, behøver testgrundlaget ikke at være færdigdefineret (se afsnit 1.4.1). I agil softwareudvikling bliver denne reviewcyklus iterativ, da de pågældende userstories raffineres i begyndelsen af hver iteration

- Der findes godkendt budget og tidsplan for testobjektets resterende opgaver

Testbetingelser er typisk identificeret ved en analyse af testgrundlaget sammen med testformålene (som beskrevet i testplanlægningen). I situationer, hvor dokumenterne kan være forældede eller ikke findes, kan testbetingelserne identificeres ved samtale med de relevante interessenter (f.eks. på workshops eller i løbet af den iterative planlægning). I agil softwareudvikling bliver acceptkriterierne, der defineres som en del af userstories, ofte anvendt som grundlag for testdesignet.

Selvom testbetingelserne generelt er tilpasset det objekt, der skal testes, opstår der visse standardovervejelser for testanalytikeren.

- Det kan generelt anbefales at definere testbetingelserne på forskellige detaljeniveauer. Først kan man identificere betingelser for testen på højt niveau, f.eks. "funktionalitet for skærm x". Efterfølgende kan man identificere mere detaljerede betingelser som grundlag for specifikke testcases, f.eks. "skærm x afviser et kontonummer, der mangler et ciffer for at være den korrekte længde". Ved at anvende denne slags hierarkiske tilgang til at definere testbetingelserne, kan det hjælpe med at sikre, at der er tilstrækkelig dækning på højt niveau. Det gør, at en testanalytiker kan arbejde med betingelser på højt niveau for userstories, der ikke er blevet raffineret endnu
- Hvis produktrisiciene er blevet defineret, bør nødvendige testbetingelser for hver produktrisiko identificeres og spores tilbage til risikofaktor

Anvendelsen af testteknikker (som identificeret inden for teststrategien og/eller testplanen) kan være nyttigt i testanalyseprocessen og kan anvendes til at understøtte de følgende formål:

- Identificere testbetingelser
- Mindske sandsynligheden for at komme til at udelade testbetingelser
- Definere mere præcise og nøjagtige testbetingelser

Efter testbetingelserne er blevet identificeret og raffineret, kan gennemgang af betingelserne udføres med interessenterne for at sikre, at kravene er forstået korrekt, og at testene følger projektets mål.

Ved afslutning af testanalysen for et givet område (f.eks. en specifik funktion) bør testanalytikeren vide, hvilke specifikke tests der skal designes til det område.

1.4 Testdesign

Ved at overholde det omfang, der er besluttet under testplanlægningen, fortsætter testprocessen med, at testanalytikeren designer de tests, der skal implementeres og afvikles. Testdesignet omfatter følgende aktiviteter:

- Afgøre i hvilke testområder det er mest passende at bruge henholdsvis lavniveau og højniveau testcases
- Afgøre hvilke(n) testteknik(ker), der muliggør, at den nødvendige dækning kan opnås. I testplanlægningen besluttes det, hvilke teknikker der kan anvendes
- Anvende testteknikker til at designe testcases og sæt af testcases, der dækker de identificerede testbetingelser
- Identifikation af nødvendige testdata for at understøtte testbetingelser og testcases
- Design af testmiljøet og identifikation af nødvendig infrastruktur, såvel som værktøjer
- Sørge for tovejs sporbarhed (f.eks. mellem testgrundlag, testbetingelser og testcases)

I løbet af risikoanalyse og testplanlægning kan man identificere de kriterier, der skal bruges til prioritere testprocessen fra analyse og design til implementering og afvikling.

Afhængigt af de testtyper, der skal designes, kan et af startkriterierne være tilgængelighed af værktøjer, der skal anvendes i løbet af designarbejdet.

I løbet af testdesignet bør testanalytikeren som minimum overveje følgende:

- For visse typer af testelementer bør man kun definere testbetingelser i stedet for at gå i detaljer med at definere testscripts med de påkrævede sekvenser af instruktioner, dertil skal for at udføre tests. I sådanne tilfælde bør testbetingelserne blot defineres, så de kan anvendes som vejledning for tests uden manuskript
- Kriterier for at bestå/fejle bør beskrives tydeligt
- Tests bør designes, så de er let forståelige for andre testere end forfatteren. Hvis ikke forfatteren selv skal udføre hele testen, vil andre testere skulle læse og forstå de tidligere beskrevne tests for at forstå testformålene og testens relative vigtighed
- Tests skal også være forståelige for andre interessenter f.eks. udviklere, der vil gennemgå tests, og auditører, der skal godkende testene
- Tests bør dække alle typer interaktioner med testobjektet og bør ikke begrænses til menneskers interaktioner igennem user-visible brugergrænsefladen. De kan f.eks. også omfatte interaktion med andre systemer og tekniske eller fysiske begivenheder (se [IREB_CPRE] for yderligere detaljer)
- Tests bør designes til at teste grænseflader mellem forskellige testobjekter, såvel som objekternes egen adfærd
- Testdesign bør prioriteres og balanceres, så de passer til risikoniveau og forretnings værdier

1.4.1 Testcases på lavt og højt niveau

En af testanalytikerens opgaver er at afgøre det bedste designniveau for testcases i en særlig situation. Lavniveau og højniveau testcases dækkes i [ISTQB_FL_SYL]. Nogle fordele og ulemper ved at anvende disse beskrives i de følgende lister:

Lavniveau testcases byder på følgende fordele:

- Uerfarne testmedarbejdere kan anvende de detaljerede oplysninger, der leveres inden for projektet. Lavniveau testcases leverer alle de specifikke oplysninger og procedurer, der er nødvendige for, at testeren kan afvikle testcasen (herunder datakrav) og verificere resultaterne
- Tests kan afvikles på ny af forskellige testere og bør opnå de samme testresultater
- Ikke-umiddelbare defekter i testgrundlaget kan blive afsløret
- Detaljeniveauet tillader en uafhængig verifikation af testene, f.eks. audits om nødvendigt
- Den tid, der bruges på automatiseret implementering af testcases kan reduceres

Lavniveau testcases har følgende ulemper:

- De kan kræve en betydelig indsats både i forhold til oprettelse og vedligeholdelse
- De vil ofte begrænse testerens opfindsomhed i løbet af afviklingen
- De kræver, at testgrundlaget er veldefineret
- Deres sporbarhed til testbetingelser kan kræve en større indsats end ved højniveau testcases

Højniveau testcases giver følgende fordele:

- De udstikker retningslinjer for det, der skal testes, og lader testanalytikeren variere de faktiske data eller den procedure, der følges, når testen afvikles
- De kan give bedre risikodækning end lavniveau testcases, fordi de i nogen grad varierer, hver gang de afvikles
- De kan defineres tidligt i kravsprocessen
- De anvender testanalytikerens erfaringer med tests og testobjekter, når testen afvikles
- De kan defineres, når der ikke kræves detaljerede og formelle dokumenter
- De er bedre egnet til genanvendelse i forskellige testcyklusser, når man kan bruge forskelligartede testdata

Højniveau testcases har følgende ulemper:

- De er sværere at reproducere, hvilket gør verificering mere vanskelig. Det er fordi de mangler de detaljerede beskrivelser, der findes i lavniveau testcases
- Der kan muligvis kræves mere erfaren testpersonale til at afvikle dem
- Når der automatiseres på baggrund af højniveau testcases, kan manglende detaljer gøre, at det er de forkerte faktiske resultater, der bliver valideret, eller at elementer, der skal valideres, mangler

Højniveau testcases kan anvendes til at udvikle lavniveau testcases, når kravene bliver mere veldefinerede og stabile. I dette tilfælde oprettes testcasen sekventielt med et flow fra højt niveau til lavt niveau, hvor kun lavniveau testcases anvendes til afvikling.

1.4.2 Design af testcases

Testcases designes via trinvis udarbejdelse og raffinering af de identificerede testbetingelser vha. testteknikker (se kapitel 3). Testcases bør kunne gentages, verificeres og spores tilbage til testgrundlaget (f.eks. krav).

Testdesign omfatter identificering af det følgende:

- Formål (dvs. de iagttagelige, målbare formål i testafviklingen)
- Forudsætninger som f.eks. projektet eller de lokaliserede krav til testmiljøet og leveringsplanerne, systemets tilstand inden testafviklingen osv.
- Testdatakrav (både testcasens inputdata, såvel som data, der skal eksistere i systemet, for at testcasen kan afvikles)
- Forventede resultater med eksplicitte kriterier for at bestå/fejle
- Slutbetingelser som f.eks. påvirkede data, systemets tilstand efter testafviklingen, udløsende faktorer for videre behandling osv.

En given udfordring kan udgøre definitionen af testens forventede resultat. Manuel beregning kan ofte være triviel og med risiko for at man begår fejl; hvis det er muligt, er det bedre at finde eller oprette et automatiseret testorakel. Når det forventede resultat skal identificeres, skal testerne ikke kun tænke på output på skærmen, men også på data og slutbetingelser for testmiljøet. Hvis testgrundlaget er tydeligt defineret, burde det teoretisk set være enkelt at identificere det korrekte resultat. Men testgrundlagets dokumentation kan være ukonkret, selvmodsigende, mangle dækning af nøgleområder eller i det hele taget mangle. I sådanne tilfælde skal en testanalytiker have erfaring med emnet eller adgang til det. Og selvom testgrundlaget er velbeskrevet, kan komplekse interaktioner mellem komplekse stimuli og responser gøre definitionen af de forventede resultater vanskelig; derfor er det nødvendigt med et testorakel. I agil softwareudvikling kan testoraklet være produktets ejer. Afvikling af testcases uden en måde at afgøre resultaternes validitet tilføjer ikke megen værdi og kan ofte generere ugyldige testrapporter eller tillid til systemet på et fejlagtigt grundlag.

De aktiviteter, der beskrives ovenfor kan anvendes på alle af testens niveauer, selvom testgrundlaget varierer. Når man analyserer og designer tests, er det vigtigt at huske hvilket niveau testen sigter mod, såvel som testens formål. Det kan hjælpe med at afgøre detaljeniveauet, såvel som de nødvendige værktøjer (f.eks. drivere og stubbe på komponenttestniveauet).

Mens man arbejder med testbetingelser og testcases vil man typisk oprette en mængde dokumenter, der resulterer i testarbejdsprodukter. I praksis kan det variere betydeligt, i hvilket omfang testarbejdsprodukterne dokumenteres. Dette påvirkes af følgende faktorer:

- Projektrisici (det, der skal eller ikke skal dokumenteres)
- Den værdi, som dokumenterne tilføjer til projektet
- De standarder og/eller regler, der skal følges
- Anvendt SDLC eller tilgang (f.eks. en agil tilgang, der sigter efter "lige præcis nok" dokumentation)
- Krav til sporbarhed fra testgrundlaget, til testanalyse og videre til testdesign

Afhængigt af testens omfang adresserer testanalysen og designet testobjektet/-objekternes kvalitetsegenskaber. ISO 25010-standarden [ISO25010] udgør en brugbar reference. Når man tester hardware-/softwaresystemer, kan der gælde andre egenskaber.

Testanalysens aktiviteter og testdesign kan forbedres ved at blande dem med reviews og statiske analyser. Faktisk er udførelse af testanalyse og testdesign ofte en form for statisk test, fordi man kan finde problemer i testgrundlagets dokumenter under aktiviteten. Testanalyse og testdesign, der baseres på kravspecifikationer, er en glimrende måde at forberede sig til et reviewmøde om krav. For at kunne læse kravene og anvende dem til at oprette tests, kræver det, at testanalytikeren forstår kravene og er i stand til at afgøre, hvordan man vurderer, om de efterleves. Denne aktivitet afdækker manglende krav, krav der er uklare, ikke er testbare eller ikke har veldefinerede acceptkriterier. Ligeledes kan testarbejdsprodukter som testdata, risikoanalyser og testplanlægning indgå i reviews.

I løbet af testdesignforløbet kan man definere kravene til den detaljerede testinfrastruktur, også selvom kravene i praksis nok ikke færdiggøres inden testimplementeringen. Man må dog huske, at testinfrastrukturen omfatter mere end testobjekter og testware. F.eks. kan infrastrukturen omfatte lokaler, udstyr, personale, software, værktøjer, tilbehør, kommunikationsudstyr, autorisering af brugere og alle de andre elementer, der er påkrævet i afviklingen af testene.

Slutkriteriet for testanalysen og testdesignet kan variere meget afhængigt af projektparametre, men alle de genstande, der nævnes i disse to afsnit, bør man overveje at inkludere i de endelige slutkriterier. Det er vigtigt, at slutkriterierne er målbare, og at alle de påkrævede oplysninger til de følgende trin er tilgængelige og at alt nødvendig forberedelse er udført.

1.5 Testimplementering

Testimplementeringen forbereder den nødvendige testware til afvikling, baseret på testanalyse og -design. Den omfatter de følgende aktiviteter:

- At udvikle testprocedurer og om muligt at udvikle automatiserede testscripts
- At organisere testprocedurer og automatiserede testscripts i testsuiter sådan at de kan afvikles under et testforløb
- At konsultere testmanageren om at prioritere afviklingen af testcases og testsuiter
- At oprette en testafviklingsplan, herunder resursetildeling, så man kan begynde afvikling af testcases (se [ISTQB_FL_SYL] afsnit 5.2.4)
- At færdiggøre forberedelsen af testdata og testmiljøer
- At opdatere sporbarheden mellem testgrundlaget og testware f.eks. testbetingelser, testcases, testprocedurer, testscripts og testsuites

I løbet af testimplementeringen skal testanalytikerne identificere en optimal rækkefølge for afviklingen af testcases og samle dem i testprocedurer. For at kunne definere testprocedurerne kræver det at man identificerer begrænsninger og afhængigheder, der kan påvirke rækkefølgen af testcases. Testprocedurerne dokumenterer de tidlige forudsætninger (f.eks. indlæsning af testdata fra dataarkivet) og andre aktiviteter.

Testanalytikeren identificerer testprocedurer og automatiserede testscripts der kan grupperes (f.eks. de tester alle en bestemt højniveauforretningsproces) og organiseres i testsuits. Dette gør det muligt at afvikle relaterede testcases samtidig.

Testanalytikeren arrangerer testsuits inden for en testafviklingsplan på en måde der resulterer i effektiv testafvikling. Hvis der anvendes risikobaseret teststrategi, vil risikoniveauet være den primære faktor til at afgøre rækkefølgen for afvikling af testcases. Der kan også være andre faktorer, der afgør rækkefølgen, f.eks. tilgængeligheden af de rette personer, udstyr, data og de funktioner, der skal testes.

Softwaren bliver ofte tilgængelig i etaper, og testen skal koordineres i forhold til den rækkefølge, som softwaren bliver tilgængelig i. Særligt i iterativ og inkrementel udvikling er det vigtigt, at testanalytikeren koordinerer med udviklingsteamet for at sikre, at softwaren bliver releaset til test i en testbar rækkefølge.

Detaljeniveauet og den tilknyttede kompleksitet for det arbejde, der udføres i løbet af testimplementeringen, kan påvirkes af detaljerne i testbetingelserne og testcases. I nogle situationer kan man være underlagt særlige regler, og det bør kunne bevises, at testarbejdsprodukterne følger de gældende standarder f.eks. den amerikanske standard DO- 178C (i Europa, ED 12C). [RTCA DO-178C/ED-12C].

Som nævnt ovenfor kræves der testdata for de fleste tests, og i nogle tilfælde kan datasættene være meget omfattende. I løbet af implementeringen skal testanalytikerne oprette input- og miljødata, der skal indlæses i databaser og lignende. Disse data skal være "fit for purpose", så det er muligt at spore defekter. Testanalytikere kan eventuelt også selv oprette data til datadrevne og nøgleordsdrevne tests (se afsnit 6.2), såvel som til manuelle tests.

Testimplementering handler også om testmiljøet eller -miljøerne. I løbet af testimplementeringen skal testmiljøerne være sat helt op og verificeret inden testafviklingen kan begynde. Et "fit for purpose" testmiljø er essentielt, dvs. testmiljøet skal være i stand til at afsløre eksisterende defekter i løbet af den kontrollerede test, operere normalt, når der ikke er nogen fejl, og om nødvendigt kunne gengive produktionen eller slutbrugermiljøet for højniveaustest i en tilstrækkelig grad. Det kan være nødvendigt at ændre i testmiljøerne i løbet af testafviklingen, hvis der sker uventede ændringer, opstår uventede testresultater eller ved andre påvirkninger. Hvis ændringer i testmiljøet opstår under afviklingen, er det vigtigt at vurdere ændringernes indvirkning på de tests, der allerede er afviklet.

Under testimplementeringen skal testanalytikerne vide hvem der har ansvar for at oprette og vedligeholde testmiljøet og sørge for at have adgang til dem. Al testwaren, testsupportværktøjer og tilknyttede processer skal være klar til brug. Det omfatter håndtering af konfigurationen, håndtering af defekter og testlogning og -styring. Desuden skal testanalytikerne godkende de procedurer, der indsamler data til evaluering af den nuværende status i forhold til rapportering af slutkriterier og resultater.

Det vil være fornuftigt at anvende en balanceret tilgang til den testimplementering, man beslutter sig for i løbet af testplanlægningen. F.eks. bliver risikobaserede analytiske teststrategier ofte kombineret med reaktive teststrategier. I dette tilfælde skal en vis procentdel af testimplementeringsindsatsen tildeles tests, der ikke følger et fastlagt script (ikke-scripted).

Unscripted tests bør ikke være tilfældige eller uden formål, da de kan være uforudsigelige i forhold til varighed og omfang, og fordi de kun finder få defekter. De bør i stedet udføres i tidsafgrænsede sessioner, der hver især guides af et testcharter, men med friheden til at afvige fra charterets forskrifter, hvis det kan lede til mere produktive testmuligheder, der opdages i løbet af sessionen. Med tiden udvikler testere en række erfaringsbaserede testteknikker, så som angreb [Whittaker03], fejlgætning [Myers11] og udforskende tests [Whittaker09]. Testanalyse, testdesign og testimplementering finder stadig sted, men de finder primært sted i løbet af testafviklingen.

Når man følger den slags reaktive teststrategier, påvirker resultatet analyse, design og implementering af efterfølgende tests. Selvom disse strategier er lette og ofte er effektive til at finde defekter, kan der være ulemper, herunder følgende:

- Der kræves ekspertise fra testanalytikere
- Varigheden kan være svær at forudsige
- Det kan være svært at spore testdækningen
- Man kan miste muligheden for at kunne gentage en test, hvis ikke man sørger for at dokumentere testen eller har en god værktøjsunderstøttelse

1.6 Testafvikling

Testafviklingen udføres i henhold til testafviklingsplanen og indeholder følgende opgaver: (se [ISTQB_FL_SYL])

- At afvikle manuelle tests, herunder udforskende tests
- At afvikle automatiserede tests
- At sammenligne de faktiske resultater med de forventede resultater
- At analysere anomalier for at afgøre deres mest sandsynlige årsager
- At rapportere defekter på baggrund af observerede afvigelser
- At logge testafviklingens faktiske resultater
- At opdatere sporbarheden mellem testgrundlaget og testwaren for at afgøre testresultaterne
- At afvikle regressionstests

Testafviklingens opgaven kan enten udføres af testere eller testanalytikere.

Yderligere, typiske opgaver der udføres af testanalytikere:

- At genkende klynger af defekter, der kan indikere et behov for yderligere test af bestemte dele af softwaren
- At foreslå kommende, udforskende tests på grundlag af fund fra eksisterende, eksplorative tests
- At identificere nye risici fra oplysninger, der er opnået ved testafvikling
- At foreslå forbedringer af ethvert arbejdsprodukt ud fra testimplementeringen (f.eks. forbedringer af testprocedurer)

2. Testanalytikerens opgaver i risikobaserede tests — 60 minutter

Nøgleord

produktrisiko, risikobaseret test, risikoidentifikation, risikominimering

Læringsmål for testanalytikerens opgaver i løbet af testprocessen

2.1 Testanalytikerens opgaver i risikobaseret test

TA-2.1.1 (K3) Deltage i risikoidentifikation, udfører risikovurdering og foreslå passende risikominimering i en given situation

2.1 Introduktion

Testmanagere har ofte det overordnede ansvar for at etablere og håndtere en risikobaseret teststrategi. De kan typisk få en testanalytiker til at implementere den risikobaserede tilgang.

Testanalytikere bør være aktivt involveret i de følgende risikobaserede testopgaver:

- Risikoidentifikation
- Risikovurdering
- Risikominimering

Testanalytikeren skal udføre disse opgaver iterativt i løbet af SDLC-forløbet for at håndtere opstående risici, så der kan omprioriteres og for regelmæssigt at evaluere og kommunikere risikostatussen (se [vanVeenendaal12] og [Black02] for yderligere detaljer). I agil softwareudvikling er de tre opgaver ofte forbundet i en såkaldt risikosession med fokus på en iteration eller en release.

Testanalytikere bør arbejde inden for den risikobaserede testramme, der er oprettet af testmanageren til projektet. De bør bidrage med deres viden om forretningsdomæne risici, der er indlejret i projektet som f.eks. de risici, der er forbundet med funktionel sikkerhed, forretningsmæssige og økonomiske overvejelser, såvel som politiske faktorer og andet.

2.2 Risikoidentifikation

Ved at trække på det bredest mulige udvalg af interessenter, vil risikoidentifikationen have større held med at spore det størst mulige antal af relevante risici.

Testanalytikere besidder ofte en unik viden vedrørende det system, der testes inden for det specifikke forretningsdomæne. Det betyder, at de er godt rustet til de følgende opgaver:

- At udføre interviews med domæneeksperter og -brugere
- At udføre uafhængige vurderinger
- At anvende risikoskabeloner
- At deltage i risiko-workshops
- At deltage i brainstorming-sessioner med potentielle og nuværende brugere
- At definere testtjeklister
- At indhente tidligere erfaringer med lignende systemer eller projekter

Testanalytikere bør arbejde tæt sammen med brugere og andre domæneeksperter (f.eks. requirement engineers og forretningsanalytikere) for at afgøre de områder for forretningsrisici, der skal behandles i løbet af testforløbet. I agil softwareudvikling tillader det tætte forhold mellem interessenterne, at risikoidentifikation sker jævnligt, f.eks. ved iterationsplanlægningsmøder.

Eksempler på risici, der kan identificeres i et projekt omfatter:

- Problemer med funktioner, f.eks. ukorrekte beregninger
- Brugervenlighedsproblemer, f.eks. en utilstrækkelig mængde genvejstaster
- Flytbarhedsproblemer, f.eks. ikke at kunne installere en applikation på en bestemt platform

2.3 Risikovurdering

Hvor risikoidentifikation drejer sig om at identificere så mange relevante risici som muligt, drejer risikovurdering om at studere de identificerede risici, især at kategorisere hver risiko og afgøre deres risikoniveau.

For at afgøre risikoniveauet, skal man typisk vurdere sandsynligheden for at risikoen sker, og effekten af risikoen. Sandsynligheden, for at risikoen sker, fortolkes generelt som sandsynligheden for, at det potentielle problem kan eksistere i systemet i løbet af testen og kan findes, når systemet er i produktion. Tekniske testanalytikere skal bidrage til at finde og forstå den potentielle sandsynlighed for hver risikofaktor, hvorimod testanalytikere bidrager til forståelsen af den potentielle effekt, det kan have for forretningen, hvis problemet opstår (i agil softwareudvikling, kan denne opdeling af roller være mindre udtalt).

Risikoens effekten fortolkes som regel på grundlag af hvor alvorlig en effekt er for brugerne, kunderne og andre interessenter. Med andre ord opstår den fra forretningsrisici. Testanalytikere bør bidrage til at identificere og vurdere den potentielle effekt i forretningsdomænet eller for brugeren af hver risikofaktor. Faktorer, der har indflydelse på forretningen, kan omfatte følgende:

- Anvendelsesfrekvens for hver påvirket funktion
- Forretningens potentielle tab
- Finansielle skader
- Økologiske eller sociale tab eller ansvar
- Civile eller lovmæssige sanktioner
- Funktionelle sikkerhedsmæssige hensyn
- Bøder, licensfratagelse
- Mangel på rimelige alternativer, hvis folk ikke kan fortsætte arbejdet
- Funktionens synlighed
- Synlighed af afvigelse, der leder til negativ omtale og potentiel skade for forretningens image
- Tab af kunder

Testanalytikerne skal fastslå forretningens risikoniveau på grundlag af retningslinjer fra testmanageren og tilgængelige risikooplysninger. Disse kan klassificeres ved hjælp af en skala (f.eks. numeriske tal eller lav / middel / høj) eller trafiksignalfarver. Når risikosandsynligheden og effekten af risiko er tildelt, bruger testmanagere disse værdier til at bestemme risikoniveauet for hver risiko. Dette risikoniveau bruges derefter til at prioritere risikoreducerende aktiviteter [vanVeenendaal12].

2.4 Risikominimering

I løbet af projektet skal testanalytikerne sørge for:

- At reducere produktrisikoen ved at designe effektive testcases, der på utvetydig vis demonstrerer, om tests er bestået eller fejler og ved at deltage i reviews af softwarearbejdsprodukter som krav, design og brugerdokumenteren
- At implementere passende aktiviteter for reducering af risici som beskrevet i teststrategien og testplanen (f.eks. at teste en særligt højrisiko forretningsproces vha. bestemte testteknikker)
- At revurdere kendte risici baseret på de yderligere oplysninger, der indsamles i løbet af projektet, og justere sandsynlighed for risiko og/eller effekten af risiko hvor det er relevant
- At identificere nye risici ud fra de oplysninger, der er indhentet via testen

Når man taler om produktrisici, udgør test et essentielt bidrag til reducering af sådanne risici. Testerne reducerer risici ved at finde defekter og gøre opmærksom på mulighederne for at afhjælpe defekterne inden release. Hvis testerne ikke finder defekter, reducerer tests risikoen ved at bidrage med beviser på, at systemet fungerer korrekt under visse omstændigheder (dvs. de testede betingelser). Testanalytikere hjælper med at afgøre potentialet for risikominimering ved at undersøge muligheder for at indsamle præcise testdata, oprette og teste realistiske brugerscenarier og udføre eller føre tilsyn med brugervenlighedsstudier m.m.

2.4.1 Prioritering af tests

Risikoniveauet anvendes også til at prioritere tests. En testanalytiker kan også beslutte, at der er en høj risiko i et område af transaktionsnøjagtighed i et regnskabssystem. For at reducere risikoen kan testeren arbejde sammen med andre forretningsdomæneeksperter om at samle et stærkt datasæt, der kan behandles og verificeres for nøjagtighed. Ligeledes kan en testanalytiker afgøre, at brugervenlighedsproblemer udgør en

betydelig risiko for et nyt testobjekt. I stedet for at vente på en brugeraccepttest til at opdage mulige problemer, kan testanalytikeren prioritere en tidlig brugervenlighedstest baseret på en prototype. En sådan test vil hjælpe med at identificere og løse designproblemer i forhold til brugervenlighed langt tidligere end en brugeraccepttest. Denne prioritering skal overvejes så tidligt som muligt i planlægningsstadiet, så der er plads til test på de nødvendige tidspunkter i tidsplanen.

I nogle tilfælde afvikles alle tests med højeste risiko først (dette kaldes "depth-first" eller ”dybde først”); i andre tilfælde anvendes der en tilgang, hvor man udvælger eksempler på tests ud fra alle identificerede risici ved at anvende risikoniveauer til at vægte valget, samtidig med, at der sikres dækning mindst én gang for hvert risici (dette kaldes "breadth-first" eller ”bredde først”).

Uanset om tilgangen til den risikobaserede test er depth-first eller breadth-first, er det muligt, at den planlagte tid er gået uden alle testene, er blevet afviklet. Risikobaseret tests lader testerne rapportere til ledelsen om det tilbageværende risikoniveau på dette tidspunkt, og lader ledelsen beslutte, om testforløbet skal forlænges, eller om de tilbageværende risikotilfælde skal overføres til andre brugere, kunder, helpdesken/teknisk support og/eller driftspersonalet.

2.4.2 Justering af testen til fremtidige testcyklusser

Risikovurdering er ikke en engangsaktivitet, der skal udføres inden begyndelsen af testimplementeringen; det er en vedvarende proces. Hver fremtidig planlagt testcyklus bør indgå i en ny risikoanalyse, så faktorer som disse kan medregnes:

- Produktrisici med nye eller betydelige ændringer
- Områder, der er ustabile eller har tilbøjelighed til at vise afvigelser i test
- Risici fra de rettede defekter
- Typiske defekter, der er fundet i test
- Områder, der ikke er tilstrækkeligt testet (lav kravdækning)

3. Testteknikker 630 minutter

Nøgleord

beslutningstabeltest, black-box testteknik, defektbaseret testteknik, defektklassifikationssystem, erfaringsbaseret test, erfaringsbaseret testteknik, fejlgætning, grænseværdianalyse, klassifikationstræteknik, parvis test, testcharter, tilstandsovergangstest, tjeklistebaseret test, udforskende test, usecasetest, ækvivalenspartitionering

Læringsmål for testteknikker

3.1 Introduktion

Ingen læringsmål

3.2 Black-box testteknikker

TA-3.2.1 (K4) Analyser givne specifikationer og design testcases ved at anvende ækvivalenspartitionering
TA-3.2.2 (K4) Analyser givne specifikationer og design testcases ved at anvende grænseværdianalyse
TA-3.2.3 (K4) Analyser givne specifikationer og design testcases ved at anvende beslutningstabeltest
TA-3.2.4 (K4) Analyser givne specifikationer og design testcases ved at anvende tilstandsovergangstest
TA-3.2.5 (K2) Forklar, hvordan klassifikationstrædiagrammer understøtter testteknikkerne
TA-3.2.6 (K4) Analyser givne specifikationer og design testcases ved at anvende parvis test
TA-3.2.7 (K4) Analyser givne specifikationer og design testcases ved at anvende casetests
TA-3.2.8 (K4) Analyser et system eller dets kravsspecifikation for at afgøre de sandsynlige typer af defekter, man kan finde, og vælg de passende black-box testteknik(ker)

3.3 Erfaringsbaserede testteknikker

TA-3.3.1 (K2) Forklar principperne for erfaringsbaserede testteknikker og fordele og ulemper sammenlignet med black-box og defektbaserede testteknikker
TA-3.3.2 (K3) Identificer udforskende tests ud fra et givet scenarie
TA-3.3.3 (K2) Beskriv anvendelsen af defektbaserede testteknikker, og forklar forskellene ved at anvende dem i stedet for black-box testteknikker

3.4 Anvendelse af de mest passende testteknikker

TA-3.4.1 (K4) For et givet projekt, afgør da, hvilke black-box eller erfaringsbaserede testteknikker, der skal anvendes for at nå de specifikke målsætninger

3.1 Introduktion

De testteknikker, der diskuteres i dette kapitel, er opdelt i de følgende kategorier:

- Black-box
- Erfaringsbaseret

Disse teknikker er komplementære og kan anvendes, hvor det er passende i enhver test eller aktivitet uanset det testniveau, der udføres.

Bemærk, at begge kategorier af teknikker kan anvendes til at teste funktionelle og ikke-funktionelle kvalitetsegenskaber. Testsoftware egenskaber diskuteres i næste kapitel.

De testteknikker, der diskuteres i disse afsnit, vil primært fokusere på, hvordan man afgør, hvilke testdata er mest optimale (f.eks. ud fra ækvivalenspartitionering) eller afleder tests f.eks. fra tilstandsmodeller). Normalvis kombinerer man teknikker, når man opretter hele testcases.

3.2 Black-box testteknikker

Black-box testteknikker introduceres i ISTQB® Foundation Level Syllabus [ISTQB_FL_SYL].

Black-box testteknikkernes almene karakteristika omfatter:

- Modeller - som f.eks. tilstandsovergangsdiagrammer og beslutningstabeller - oprettes i løbet af testdesign i overensstemmelse med testteknikken
- Testbetingelserne udledes systematisk fra disse modeller

Testteknikkerne indeholder generelt de dækningsbetingelser, der kan anvendes til at måle testdesignet og testafviklingen. Selvom man opfylder dækningskriterierne fuldstændigt, betyder det ikke, at hele sættet af tests er fuldt dækkende. I stedet viser det at modellen ikke kræver yderligere testdækning baseret på denne teknik.

Black-box test er generelt baseret på en form for specifikationsdokument f.eks. en udspecificering af systemkrav eller userstories. Da specifikationsdokumentet bør beskrive systemets adfærd, særligt i henhold til funktionel egnethed, vil det at udlede tests ud fra kravene ofte være en del af processen, når man tester systemets adfærd. I nogle tilfælde vil der ikke være nogen specifikationsdokumenter, men der vil stadig være implicitte krav, f.eks. at skulle erstatte et legacy systems funktionel egnethed.

Der findes en række black-box testteknikker. Disse teknikker er rettet imod forskellige typer software og scenarier. De nedenstående afsnit viser hver tekniks anvendelighed, og nogle begrænsninger og udfordringer, som testanalytikeren kan opleve, såvel som den metode, dækning måles med, og de typer af defekter, man leder efter.

Se venligst [ISO29119-4], [Bath14], [Beizer95], [Black07], [Black09], [Copeland04], [Craig02], [Forgács19], [Koomen06] og [Myers11] for yderligere detaljer.

3.2.1 Ækvivalenspartitionering

Ækvivalenspartitionering (EP) er en teknik, der anvendes til at reducere det antal af testcases, der kræves for på effektiv vis at teste håndteringen af input, output, interne værdier og tidsrelaterede værdier. Partitionering anvendes til at oprette ækvivalenspartitioner (de kaldes ofte for ækvivalensklasser), der oprettes ud fra et sæt af værdier, som skal behandles på samme måde. Ved at vælge en repræsentativ værdi ud fra en partition, formodes det, at der er dækning for alle elementer i samme partition.

Normalt bestemmer flere parametre testobjektets opførsel. Forskellige teknikker kan anvendes til at kombinere ækvivalenspartitioner fra forskellige parametre til testcases.

Anvendelighed
Denne teknik er anvendelig på alle testniveauer og er passende, når alle dele i et sæt af værdier, der skal testes skal behandles på samme måde, og hvor sættet af værdier, der anvendes i applikationen, ikke interagerer. En ækvivalenspartition kan være et hvilket som helst ikke-tomt sæt af værdier, fx: ordnet, uordnet, diskret, kontinuerlig, uendelig, endelig eller endda en singleton. Det udvalgte sæt af værdier er anvendeligt for både gyldige og ugyldige partitioner (f.eks. partitioner, der indeholder værdier, som bør regnes for ugyldige til den software, der testes).

ÆP er stærkest, når teknikken anvendes sammen med en grænseværdianalyse, der udvider testværdierne til at inkludere de værdier, der ligger inden for partitionens grænseområder. En ÆP, der anvender værdier fra de gyldige partitioner, er en ofte anvendt teknik til at smoketeste et nyt build eller en ny release, da den hurtigt kan afgøre, om de basale funktioner virker.

Begrænsninger/Udfordringer
Hvis formodningen er ukorrekt, og værdierne i partitionen ikke skal håndteres helt ens, kan denne teknik lede til, at man overser defekter. Det er vigtigt at vælge partitionerne med omhu. F.eks. ville det være bedre, hvis et inputfelt, der accepterer positive og negative tal, bliver testet som to separate, gyldige partitioner - et til positive tal og et til negative tal - da det er sandsynligt, at de skal behandles forskelligt. Afhængigt af om nul er tilladt, kan man også oprette en ekstra partition for at teste dette. Det er vigtigt for en testanalytiker at forstå den underliggende databehandling for at afgøre den bedste form for partitionering af værdierne. Dette kan kræve hjælp til forståelse af kodens design.

Testanalytikeren skal også tage højde for mulige afhængigheder mellem ækvivalenspartitioner af forskellige parametre. F.eks. i et flyreservationssystem kan parameteren "ledsagende voksen" kun bruges i kombination med aldersklassen "barn".

Dækning
Dækningens omfang afgøres ved at tage antallet af partitioner, hvor man har testet en værdi, og dele tallet med det antal af partioner, som er blevet identificeret. ÆP-dækningen skal så skrives i procenter. At anvende flere værdier for en enkelt partition øger ikke dækningsprocenten.

Hvis testobjektets opførsel afhænger af en enkelt parameter, skal hver ækvivalenspartition, uanset om den er gyldig eller ugyldig, dækkes mindst en gang.

I tilfælde af mere end én parameter skal testanalytikeren vælge en simpel eller kombinatorisk dækningstype afhængigt af risikoen [Offutt16]. Det er derfor vigtigt at skelne mellem kombinationer, der kun indeholder gyldige partitioner og kombinationer der indeholder en eller flere ugyldige partitioner. Med hensyn til kombinationerne med kun gyldige ækvivalenspartitioner er minimumskravet en simpel dækning af alle gyldige partitioner over alle parametre. Det mindste antal testcases der er nødvendige i en sådan testsuite, er lig med det største antal gyldige partitioner af en parameter, forudsat at parametrene er uafhængige af hinanden. Mere grundige dækningstyper relateret til kombinatoriske teknikker inkluderer parvis dækning (se afsnit 3.2.6 nedenfor) eller den fulde dækning af enhver kombination af gyldige partitioner. Ugyldige ækvivalenspartitioner skal testes mindst individuelt, dvs. i kombination med gyldige partitioner for de andre parametre, for at undgå defektmaskering. Så hver ugyldig partition bidrager med en testcase til testsuiten for enkel testdækning. I tilfælde af høj risiko kan yderligere kombinationer føjes til testsuiten, f.eks. hvis testsuiten består af kun ugyldige partitioner eller par med ugyldige partitioner.

Defekttyper
Testanalytikeren anvender denne teknik til at finde defekter i håndteringen af forskellige dataværdier.

3.2.2 Grænseværdianalyse

Grænseværdianalyser (BVA) anvendes til at teste den korrekte håndtering af de værdier, der ligger på grænsen af ordnede ækvivalenspartitioner. Der findes to gængse måder til at udføre en BVA: test af to-værdi eller tre-værdi grænsetest. Ved test af to-værdi vil grænseværdien (på grænsen) og værdien lige uden for grænsen (med den laveste præcision baseret på den krævede nøjagtighed) blive anvendt. F.eks. for værdier for betalinger med 2 decimaler hvis partitionen inkluderer værdierne fra 1 til 10, så vil de to testværdier i den

øvre grænse være 10 og 10,01. De nedre testgrænseværdier vil være 1 og 0,99. Grænserne defineres af de maksimale og minimale værdier i den definerede ækvivalenspartition.

For grænseværditest af tre-værdi bliver værdier før, på og over grænsen anvendt. Det tidligere eksempel omfattede de øvre grænseværditests værdierne 9,99, 10 og 10,01. De nedre grænseværditests omfattede 0,99, 1 og 1,01. Beslutningen om der skal anvendes to-værdi eller tre-værdi grænsetest, bør baseres på de risici, der er forbundet med testelementet. Tilgangen med de tre-værdi grænser anvendes til højrisikoelementer.

Anvendelighed
Teknikken er tilgængelig på alle testniveauer og kan bruges, når der findes en ækvivalenspartition med kontinuerte værdier. Derfor udføres BVA ofte sammen med ÆP-teknikken. Det kræves, at ækvivalenspartitionerne er arrangeret i en rækkefølge pga. konceptet om at ligge på eller uden for grænsen. F.eks. udgør én række af tal én partition af kontinuerte værdier. En partition, der består af tekststrenge, kan også sorteres, f.eks. efter deres leksikografiske rækkefølge, men hvis sortering ikke er relevant ud fra et forretningsmæssigt synspunkt, bør grænseværdier ikke være i fokus. Ud over grupperingen talrækker, kan grænseværdianalyser anvendes på:

- Numeriske attributer af ikke-numeriske variable (f.eks. længde)
- Løkker i afviklingscycler, herunder løkker, der findes i et tilstandsdiagram
- Elementer der bliver itereret igennem i lagrede datastrukturer så som arrays
- Størrelsen af fysiske objekter (bl.a. hukommelse)
- Længden af aktiviteter

Begrænsninger/Udfordringer
Grænseværdianalysens nøjagtighed er afhængig af at der er sket en korrekt identifikation af ækvivalenspartitioner. Derfor er den underlagt de samme begrænsninger som ækvivalenspartitioner. Testanalytikeren bør også være bevidst om præcisionen i de gyldige og ugyldige værdier for på korrekt vis at kunne afgøre, hvilke værdier der skal testes. Kun kontinuerte værdier kan anvendes til grænseværdianalyse, men partitionerne er ikke begrænset til gyldige input. F.eks. når der testes for antallet af celler i et regneark, vil der være en partition, der indeholder antallet af celler til og med det maksimale antal tilladte celler (grænsen), og en anden partition, der begynder på et celleantal med et over det maksimale (en over grænsen).

Dækning
Dækningens omfang afgøres ved at tage antallet af grænsebetingelser, der testes, og dele med antallet af identificerede grænsebetingelser (enten ved at anvende metoden for to- eller tre-værdier). Dækningen beregnes i procent.

Defekttyper
Grænseværdianalyse er pålidelig til at finde forskydninger og udeladelser af grænser og kan også finde tilfælde af ekstra grænser. Denne teknik finder defekter ved håndteringen af grænseværdier, i særdeleshed fejl ved mindre-end og større-end logik. Den kan også anvendes til at finde ikke-funktionelle defekter, f.eks. hvis et system understøtter 10.000 samtidige brugere, men ikke 10.001.

3.2.3 Beslutningstabeltests

En beslutningstabel er en tabelafbildning af et sæt betingelser og relaterede handlinger, udtrykte regler, der angiver, hvilken handling der skal forekomme for hvilket sæt betingelsesværdier [OMG-DMN]. Testanalytikere kan bruge beslutningstabeller til at analysere de regler, der gælder for softwaren under test og designe tests for at dække disse regler.

Betingelser og de resulterende handlinger af testobjektet danner rækkerne i beslutningstabellen, normalt med betingelserne øverst og handlingerne i bunden. Den første kolonne i tabellen indeholder beskrivelsen af henholdsvis betingelser og handlinger. Følgende kolonner, kaldet reglerne, indeholder henholdsvis betingelsesværdierne og de tilsvarende handlingsværdier.

Beslutningstabeller, hvor betingelserne er boolske med simple værdier "Sandt" og "Falsk" kaldes beslutningstabeller med begrænset adgang. Et eksempel på en sådan betingelse er "Brugerens indkomst <1000". Beslutningstabeller med udvidet adgang, der har flere værdier, som kan repræsentere diskrete elementer eller sæt af elementer. F.eks. kan en betingelse "Brugers indkomst" tage en af tre mulige værdier: "lavere end 1000", "mellem 1000 og 2000" og "mere end 2000".

Simple handlinger tager boolske værdier "Sandt" og "Falsk" (f.eks. Handlingen "Tilladt rabat = 20%" tager værdierne "Sandt" betegnet med "X", hvis handlingen skulle forekomme, og "Falsk" betegnes med "-" hvis ikke). Ligesom med betingelser kan handlinger også tage værdier fra andre domæner. F.eks. kan en handling "Tilladt rabat" tage en af fem mulige værdier: 0%, 10%, 20%, 35% og 50%.

Test af beslutningstabeller starter med at designe beslutningstabeller baseret på specifikationen. Regler, der indeholder umulige kombinationer af betingelsesværdier, er ekskluderet eller markeret som "uigennemførlige". Derefter skal testanalytikeren gennemgå beslutningstabellerne med de andre interessenter. Testanalytikeren skal sikre, at reglerne i tabellen er konsistente (dvs. at reglerne ikke overlapper hinanden), komplette (dvs. de indeholder en regel for hver mulig kombination af tilstandsværdier) og korrekte (dvs. de modellerer den tilsigtede adfærd).

Det grundlæggende princip i beslutningstabeltest er at reglerne danner testbetingelserne.

Når man designer en testcase der skal dække en given regel, skal testanalytikeren være opmærksom på, at inputværdierne i testcasen kan være forskellige fra betingelsens værdier i beslutningstabellen. F.eks. er den 'SANDE' værdi af betingelsen "årlig indkomst > 100.000?" kan ikke være direkte anvendelig, men kan kræve arbejde fra testeren for at definere en kundes konto med kreditter større end 100.000 i et givet regnskabsår. Tilsvarende kan det forventede resultat af testcasen være forskelligt fra handlingen i beslutningstabellen.

Når beslutningstabellen er klar, skal reglerne implementeres som testcases ved at vælge testinputværdier (og forventede resultater), der opfylder betingelserne og handlingerne.

Komprimeret beslutningstabel
Når man prøver at teste alle mulige inputkombinationer i henhold til betingelserne, kan beslutningstabeller blive meget store. En komplet beslutningstabel med begrænset adgang med ^{n} betingelser har 2^{n} regler. En teknik til systematisk reduktion af antallet af kombinationer kaldes komprimeret beslutningstabeltest [Mosley93]. Når denne teknik bruges, kan en gruppe regler med det samme sæt handlinger reduceres (komprimeres) til én regel, hvis der inden for denne gruppe er nogle betingelser der ikke er relevante for handlingen, og alle de andre betingelser forbliver uændrede. I denne resulterende regel betegnes værdierne for de irrelevante forhold som "ligeglad", normalt markeret med en bindestreg "-". For forhold med "ligeglad"-værdier kan testanalytikeren specificere vilkårlige gyldige værdier til implementering af test.

Et andet tilfælde for komprimering af regler er når en betingelsesværdi ikke kan anvendes i kombination med nogle andre betingelsesværdier, eller når to eller flere betingelser har modstridende værdier. F.eks. i en beslutningstabel for kortbetalinger, hvis betingelsen "kort er gyldigt" er falsk, er betingelsen "PIN-kode er korrekt" ikke gældende.

Komprimerede beslutningstabeller kan have meget færre regler end de fulde beslutningstabeller, hvilket resulterer i et lavere antal testcases og mindre testindsats. Hvis en given regel har "ligeglad"-poster, og kun en testcase dækker denne regel, bliver kun en af flere mulige værdier af betingelsen for denne regel testet, så en fejl, der involverer andre værdier, vil derfor muligvis ikke blive fundet. Derfor skal testanalytikeren for høje risikoniveauer i overensstemmelse med testmanageren definere separate regler for hver mulig kombination af de enkelte betingelsesværdier snarere end at skjule beslutningstabellen.

Anvendelighed
Beslutningstabel anvendes generelt til integrations-, system- og accepttestniveauer. Den kan også gælde for komponenttests, når en komponent er ansvarlig for et sæt af beslutningslogikker. Teknikken er særligt anvendelig, når testobjektet er specificeret i form af rutediagrammer eller tabeller med forretningsregler.

Beslutningstabeller er også en kravdefinitionsteknik, og nogle gange kan kravspecifikationer allerede være defineret i dette format. Testanalytikeren skal stadig deltage i review af beslutningstabeller og analysere dem før man starter med testdesignet.

Når man designer beslutningstabeller, er det vigtigt at definere rammerne og både beskrevne og ikke beskrevne regler. Generelt skal en testanalytiker være i stand til at udlede de forventede handlinger for alle beslutningsregler i overensstemmelse med specifikationer eller testoraklet.

Begrænsninger/Udfordringer
Når man overvejer kombinationer af betingelser, kan det være udfordrende at skulle finde alle de betingelser, der interagerer, særligt når kravene ikke er veldefinerede eller helt mangler. Man skal derfor være forsigtig med at vælge betingelser i et beslutningstræ, så antallet af kombinationer af betingelserne er håndterbart. I værste tilfælde vil antallet af regler stige eksponentielt.

Dækning
Den almindelige dækningsstandard for denne teknik er at dække hver regel i beslutningstabellen med en testcase. Dækningen måles som en procentdel af antallet af regler, der er omfattet af testsuiten, og det samlede antal mulige regler.

Grænseværdianalyse og ækvivalenspartitionering kan kombineres med beslutningstabelteknikken, især i tilfælde af beslutningstabeller med udvidet adgang. Hvis betingelser indeholder ækvivalenspartitioner, der er ordnet fuldstændigt, kan grænseværdierne bruges som yderligere adgang der fører til yderligere regler og testcases.

Defekttyper
Typiske defekter omfatter logikrelateret databehandling, baseret på særlige kombinationer af betingelser, der kan lede til uventede resultater. Når man laver beslutningstabeller, kan man i nogle tilfælde finde defekter i specifikationsdokumentet. Det er ikke usædvanligt at finde manglende beslutningsregler eller manglende resultater for en eller flere beslutningsregler. Det er ikke usædvanligt at forberede et sæt af betingelser og fastslå, at det forventede resultat er uspecificeret for en eller flere regler. Den mest almindelige type af defekter er udeladelser af handling (dvs. hvor der ikke findes nogen oplysning om, hvad der faktisk vil finde sted i visse situationer) og modsætninger.

3.2.4 Tilstandsovergangstests

Tilstandsovergangstest kan anvendes til at teste testobjektets evne til at skifte til eller væk fra definerede tilstande vha. gyldige overgange, såvel som at prøve at skifte til ugyldige tilstande eller dække ugyldige overgange. Hændelser gør, at testobjekter skifter fra tilstand til tilstand og udfører visse handlinger. Hændelserne kan kvalificeres af betingelser (kaldet guard conditions eller transition guards), der kan påvirke overgangsstien, som skal følges. F.eks. vil en login-hændelse med en gyldig kombination af brugernavn/kode lede til en anderledes overgang end en login-hændelse med en ugyldig kode. Denne oplysning repræsenteres i et tilstandsovergangsdiagram eller i en tilstandsovergangstabel, der også kan omfatte potentielt ugyldige overgange mellem tilstande.

Anvendelighed
Tilstandsovergangstests er også anvendelige til enhver form for software, der har definerede tilstande, og som har hændelser, der forårsager overgange mellem disse tilstande (f.eks. skærmskift). Tilstandsovergangstests kan anvendes på alle testniveauer. Indlejret software, webbaseret software og enhver slags software til overførsler er gode kandidater til denne slags test. Kontrolsystemer, f.eks. trafiklysstyresystemer, er også gode kandidater til denne slags test.

Begrænsninger/Udfordringer
At afgøre tilstandene kan ofte være den mest udfordrende del af at definere tilstandsovergangsdiagrammet eller tilstandsovergangstabellen. Hvis et testobjekt har en brugergrænseflade, bliver de forskellige skærme, der vises til brugeren, ofte repræsenteret ved tilstandene. Hvad indlejret software angår, kan tilstandene være afhængige af hardwarens tilstande.

Ud over selve tilstandene er grundenheden for tilstandsovergangstest den enkelte overgang. Man kan finde nogle typer af tilstandsovergangsdefekter ved blot at teste hver enkelt overgang, men man kan finde flere ved at teste sekvenser af overgange. En enkelt overgang kaldes en 0-switch; en sekvens med to overgange i træk kaldes for en 1-switch; en sekvens med tre overgange i træk er en 2-switch osv. Generelt repræsenterer en N-switch N + 1 overgange i træk [Chow1978]. Når N stiger, vokser antallet af N-switches meget hurtigt, hvilket gør det vanskeligt at opnå N-switch-dækning med et rimeligt, lille antal tests.

Dækning
Ligesom med de andre typer testteknikker, findes der et hierarki af niveauer for dækning. Den minimalt acceptable grad af dækning vil sige, at man har tilset alle tilstande og skiftet mellem hver overgang mindst én gang. 100% overgangsdækning (også kendt som 100% 0-switch dækning) garanterer, at hver tilstand er tilset, og at man har skiftet mellem alle overgange, medmindre systemdesignet eller tilstandsovergangsmodellen (diagram eller tabel) er defekte. Afhængigt af forholdet mellem tilstande og overgange kan det være nødvendigt at skifte mellem nogle overgange mere end en gang for at afvikle andre overgange en enkelt gang.

Begrebet "N-switch dækning" er relateret til antallet af switches, der dækker afstanden N+1 i procentdele for summen af switches i den længde. F.eks. for at opnå 100% kræver en 1-switch dækning, at alle gyldige sekvenser med to overgange i træk er blevet testet mindst en gang. Denne test kan aktivere visse typer af fejl, som en 100% 0-switch dækning ville overse.

"Tur-retur dækning" gælder de situationer, hvori overgangssekvenser danner løkker. 100% tur-retur dækning opnås ved, at alle løkker fra alle tilstande helt tilbage til den samme tilstand, der blev testet for alle tilstande, hvorfra løkken begynder og ender. Denne løkke kan ikke indeholde mere end et tilfælde af en specifik tilstand bortset fra start og stop [Offutt16].

I hver af disse tilgange vil man med en højere dækningsgrad forsøge at inkludere alle ugyldige overgange, der identificeres i tilstandsovergangstabellen. Dækningskrav og dækningssæt til tilstandsovergangstest skal identificere, om ugyldige overgange er medtaget.

Tilstandsdiagrammet eller tilstandsovergangstabellen for et testobjekt understøtter design af testcases til at opnå den ønskede dækning. Oplysningen kan også gengives i en tabel, der viser overgange mellem N-switches for en given værdi af "N" [Black09].

Man kan anvende en manuel procedure til at identificere de genstande, der skal dækkes (f.eks. overgange, tilstande eller N-switches). En af metoderne er at udskrive tilstandsovergangsdiagrammet og tilstandsovergangstabellen og bruge en kuglepen eller blyant til at markere de punkter, der dækkes, indtil den påkrævede dækning vises [Black09]. Denne tilgang ville kræve for meget tid for mere komplekse tilstandsovergangsdiagrammer og tilstandsovergangstabeller. Der bør derfor anvendes et værktøj for at understøtte tilstandsovergangstest.

Defekttyper
De følgende omfatter typiske defekter (se også [Beizer95]):

- Ukorrekt hændelsestype eller -værdi
- Ukorrekt handlingstype eller -værdi
- Ukorrekt starttilstand
- Manglende evne til at opnå sluttilstand(e)
- Manglende evne til skifte til de påkrævede tilstande
- Ekstra (unødvendige) tilstande
- Manglende evne til at udføre en gyldig overgang korrekt
- Evne til at udføre ugyldige overgange
- Forkert guard condition

Når man skal oprette en tilstandsovergangsmodel, kan det ske, at man finder defekter i specifikationsdokumentet. De mest almindelige defekttyper er udeladelser (dvs. ingen oplysninger om, hvad der bør finde sted i specifikke situationer) og modsætninger.

3.2.5 Klassifikationstræteknikken

Klassifikationstræer understøtter visse black-box testteknikker ved at muliggøre grafiske gengivelser af det datarum, man opretter, og som gælder for testobjektet.

Dataene er organiseret efter klassifikationer og klasser som i det følgende:

- Klassifikationer: Disse gengiver parametre inden for testobjektets datarum. Det kan være inputparametre, som derudover kan indeholde miljøtilstande og forudsætninger, såvel som output-parametre. F.eks. hvis en applikation kan konfigureres på mange forskellige måder, så kan klassifikationerne omfatte browser, sprog og operativsystem
- Klasser: Hver klassifikation kan indeholde ethvert antal af klasser og underklasser, der beskriver parameters hændelse. Hver klasse eller ækvivalenspartition udgør en specifik værdi inden for en klassifikation. I eksemplet ovenfor kan sprogklassifikationen omfatte ækvivalenspartition for engelsk, fransk og spansk

Klassifikationstræer lader testanalytikeren indsætte kombinationerne efter behov og ønske. Dette omfatter f.eks. kombinationer med enkeltvis, parvis (afsnit 3.2.6) og tre ad gangen.

Yderligere oplysninger vedrørende brug af klassifikationstræteknikken kan findes i [Bath14] og [Black09].

Anvendelighed
Når der oprettes et klassifikationstræ, kan det hjælpe testanalytikeren med at identificere parametre (klassifikationer) og deres ækvivalenspartitioner (klasser) som har interesse.

Yderligere analyse af klassifikationstrædiagrammet gør det muligt at identificere mulige grænseværdier. Desuden kan man identificere eller afvise visse kombinationer af input (f.eks. fordi de er inkompatible). Det resulterende klassifikationstræ kan derefter anvendes til at understøtte ækvivalenspartitioneringstests, grænseværdianalyser eller parvise tests (se afsnit 3.2.6).

Begrænsninger/Udfordringer
Når mængden af klassifikationer og/eller klasser stiger, bliver diagrammet større og sværere at anvende. Klassificeringstræteknikken opretter heller ikke komplette testcases, kun testdatakombinationer. Testanalytikeren skal levere resultaterne for hver testkombination for at skabe komplette testcases.

Dækning
Testcases kan være designet til at opnå ting som minimum klassedækning (f.eks. at alle værdier i en klassifikation testes mindst en gang). Testanalytikeren kan også beslutte sig for at dække alle kombinationer parvist eller bruge andre typer af kombinatorisk test, f.eks. tre ad gangen.

Defekttyper
De typer af defekter, som metoderne kan finde, afhænger af de teknikker, som klassifikationstræet understøtter (f.eks. ækvivalenspartitionering, grænseværdianalyse og parvis test).

3.2.6 Parvis test

Parvis test anvendes, når man tester software, hvori flere inputparametre, hver med flere mulige værdier, skal testes i kombination, og som leder til flere kombinationsmuligheder, end det giver mening at teste inden for den mulige tidsramme. Inputparametrene kan være uafhængige så alle muligheder for alle faktorer kan kombineres med alle de mulige andre faktorer (f.eks. alle valgte værdier for en inputparameter), dog er det ikke altid tilfældet (se note ved funktionsmodeller nedenfor). Kombinationen af specifikke parametre (variabel eller faktor) med en specifik værdi for en parameter kaldes for et parameterværdipar (f.eks. hvis ' farve' er en parameter med syv tilladte værdier inklusiv ’rød’, så er 'farve = rød' et parameterværdipar).

Parvis test anvender kombinatoriske teknikker for at sikre, at hvert parameterværdipar bliver testet en gang op imod hvert parameterværdipar i andre parametre (f.eks. 'alle par' iblandt parameterværdipar for to forskellige parametre bliver testet), mens man undgår at teste alle kombinationer af parameterværdipar. Hvis testanalytikeren bruger en manuel tilgang, skal der oprettes en tabel med testcases, som repræsenteres i form af rækker og en kolonne for hver parameter. Testanalytikeren skal så udfylde tabellen med værdier, så alle værdipar kan identificeres i tabellen (se [Black09]). Alle poster i tabellen, der er tomme, kan udfyldes med værdier af testanalytikeren, der anvender sit eget domænekendskab.

Der er en række værktøjer tilgængelige til at hjælpe testanalytikeren med denne opgave (se www.pairwise.org f.eks.). De kræver en liste over parametrene og deres værdier som input. Værktøjet generer et egnet sæt kombinationer af værdier for hver parameter, der dækker alle parameterværdiparrene. Værktøjets output kan anvendes som input i testcases. Bemærk at testanalytikeren skal levere det forventede resultat for hver kombination, der oprettes af værktøjerne.

Klassifikationstræer (se afsnit 3.2.5) anvendes ofte sammen med parvis test [Bath14] Nogle værktøjer understøtter design af klassifikationstræer og kan visualisere kombinationer af parametre og deres værdier visualiseres. Nogle værktøjer tilbyder parvise forbedringer. Dette hjælper til at identificere følgende oplysninger:

- Input, der skal anvendes til parvis testteknik
- Specifikke relevante kombinationer (f.eks. de ofte anvendte eller almindelige defektkilder)
- Specifikke kombinationer, der er inkompatible. Der er ingen formodning om, at de kombinerede faktorer ikke påvirker hinanden; det kan de sagtens, men de bør påvirke hinanden på acceptable måder
- Logiske forhold mellem variable. F.eks. "hvis variable1 = x, så kan variable2 ikke være y". Klassifikationstræer, der optager disse forhold, kaldes for "feature models"

Anvendelighed
Problemet med at have for mange kombinationer af parameterværdier optræder i mindst to forskellige situationer, der har med tests at gøre. Nogle testelementer involverer flere parametre, der hver især indeholder et antal mulige værdier, f.eks. en skærm med flere indtastningsfelter. I dette tilfælde udgør kombinationer af parameterværdier inputdata for testcasene. Derudover kan nogle systemer være konfigurerbare i en række dimensioner, hvilket resulterer i et potentielt set stort konfigurationsrum. I begge af disse situationer kan parvis test anvendes til at identificere delmængder af kombinationer der er håndterbare.

For parametre med mange værdier kan ækvivalenspartitionering eller en anden udvælgelsesmekanisme først anvendes individuelt på hver parameter for at reducere antallet af værdier for hver parameter, inden parvis test benyttes til at reducere mængden af afledte kombinationer. Reduktionen kan f.eks. ske ved at indsætte parametrene og deres værdier i et klassifikationstræ.

Disse teknikker anvendes generelt på komponentintegrationstest, systemtest og systemintegrationstest niveauer.

Begrænsninger/Udfordringer
Den største begrænsning ved disse teknikker er formodningen om, at resultaterne af nogle få tests er repræsentative for alle tests, og at disse få tests repræsenterer den forventede anvendelse. Hvis der er opstår en uventet interaktion mellem bestemte variable, kan det blive overset i denne testteknik, hvis netop disse kombinationer ikke testes. Disse teknikker kan være svære at forklare til folk uden for fagfeltet, da de måske ikke forstår den logiske reduktion af testene. Alle sådanne forklaringer bør balanceres ved at nævne resultatet fra de empiriske studier [Kuhn16], der viste, at inden for forsøg med medicinsk udstyr, så blev 66% af alle fejl udløst af en enkelt variabel, og 97% af enten en eller to variable, der interagerer. Der er en iboende risiko for, at parvis testteknik ikke kan spore systemfejl, hvor tre eller flere variable interagerer.

Det kan være vanskeligt at identificere parametrene og deres respektive værdier. Denne opgave bør derfor udføres med understøttelse af klassifikationstræer, hvor end det er muligt (se afsnit 3.2.5). Desuden kan det være svært at finde det minimale antal kombinationer for de forskellige dækningsniveauer med manuelle

metoder. Værktøjer kan bruges til at finder det mindst mulige sæt af kombinationer. Nogle værktøjer understøtter evnen til at tvinge visse kombinationer til at blive inkluderet eller ekskluderet fra det endelige udvalg af kombinationer. Denne mulighed kan anvendes af en testanalytiker til at fremhæve eller fjerne fokus fra faktorer baseret på domænekendskab eller oplysninger om produktanvendelse.

Dækning
100% parvis dækning kræver, at hvert værdipar i alle par af parametre inkluderes i mindst en kombination.

Defekttyper
De mest almindelige defekttyper, der kan findes med denne testteknik, er defekter, som er relateret til den kombinerede værdi af to parametre.

3.2.7 Usecasetests

Usecasetests giver transaktionelle og scenariebaserede tests, der bør efterligne anvendelsen af en komponent eller et system, som beskrevet af usecasen. Usecases defineres i forhold til interaktioner mellem aktørerne og en komponent eller et system, der skal udføre en målsat handling. Aktører kan være menneskelige brugere, ekstern hardware eller andre komponenter eller systemer.

Anvendelighed
Usecasetests anvendes generelt i system- og accepttests. De kan også anvendes i integrationstests, hvis komponenternes eller systemernes adfærd beskrives i usecases. Usecases er også ofte grundlaget for performancetest, fordi de gengiver en realistisk anvendelse af systemet. Scenarierne, der står beskrevet i usecases kan tildeles virtuelle brugere til at skabe en realistisk belastning af systemet (så længe belastningen og performancekrav står beskrevet i usecases).

En fælles standard for use cases findes i [OMG-UML].

Begrænsninger/Udfordringer
For at være gyldige, skal usecases gengive realistiske brugertransaktioner. Usecases' specifikationer er en slags systemdesign. Kravene til det, som brugerne skal opnå, bør komme fra brugere eller brugerrepræsentanter, og de bør tjekkes efter over for de organisatoriske krav, inden man designer de tilsvarende usecases. En usecase's værdi mindskes, hvis den ikke reflekterer rigtige brugere og de organisatoriske krav, eller forhindrer dem frem for at hjælpe brugeren.

En præcis definition af undtagelser, alternativ adfærd og fejlhåndteringsadfærd er vigtig for, at dækningen bliver fyldestgørende. Usecases bør anses som vejledende, men ikke som den gennemgående definition af det, der skal testes, da det kan ske, at de ikke giver en klar definition af alle kravene. Det kan også være gavnligt at oprette andre modeller som rutediagrammer og/eller beslutningstabeller fra usecasen for at forbedre testens nøjagtighed og for at verificere selve usecasen. Ligesom for andre former for specifikation vil også en specifikation af en usecase højst sandsynligt afsløre anomalier i logikken, hvis de findes.

Dækning
Det minimale acceptable dækningsniveau i en usecase vil være at have en testcase til normal adfærd og en tilstrækkelig mængde testcases til at dække alle alternativer og al fejlhåndteringsadfærd. Hvis der kræves et minimalt testsuite, kan flere alternative typer adfærd indarbejdes i en testcase, såfremt de er gensidigt kompatible. Hvis der kræves en bedre diagnostisk kapacitet (f.eks. til at hjælpe med at isolere defekter), kan man designe en yderligere testcase per alternativ adfærd, selvom det kan være nødvendigt at samle indlejret alternativ adfærd i enkelte testcase (f.eks. afsluttende over for ikke-afsluttende alternativ adfærd inden for en "prøv igen" undtagelsesadfærd).

Defekttyper
Defekter omfatter fejlagtig håndtering af defineret adfærd, overset alternativ adfærd, ukorrekt behandling af de fremlagte betingelser og beskeder, der enten er ringe implementeret eller udført forkert.

3.2.8 Kombination af teknikker

Nogle gange kombinerer man teknikker for at oprette testcases. F.eks. kan de betingelser, der er identificeret i beslutningstabellen, udsættes for ækvivalenspartitionering for at opdage flere måder, hvorpå en betingelse kan opfyldes. I så fald vil disse testcases ikke blot dække hver eneste kombination af betingelser, men for de partitionerede betingelser, skal der blive genereret yderligere testcases, så de også dækker ækvivalens-partitioner. Når der vælges teknik, der skal anvendes, overvejes dens anvendelighed, begrænsninger og udfordringer, i forhold til testens mål, dækning og defekttyper, der ønskes fundet. Disse partitioner beskrives for hver individuel teknik i dette kapitel. Det kan ske, at der ikke findes en "bedste" teknik til situationen. Det vil ofte give den mest fyldestgørende dækning, hvis man kombinerer teknikker, hvis der er tilstrækkelig tid og ekspertise til at udføre teknikkerne.

3.3 Erfaringsbaserede testteknikker

Erfaringsbaserede tests anvender testernes kompetencer og intuition sammen med deres erfaring med lignende programmer og teknologier, når det kommer til at målrette tests til at øge opdagelse af defekter. Disse testteknikker rækker fra "hurtige tests", hvorved testerne ikke har nogen formelt forud planlagte aktiviteter at udføre, til planlagte testsessioner, der gør brug af testcharters til scriptede testsessioner. De er næsten altid brugbare, men har en særlig værdi, når de aspekter, der er inkluderet i den følgende liste af fordele, kan opnås.

Erfaringsbaserede tests har følgende fordele:

- De kan udgøre et godt alternativ til mere strukturerede tilgange i tilfælde, hvor systemdokumentationen er mangelfuld
- De kan anvendes, når testtiden er meget begrænset
- De tillader at ekspertise inden for domænet og den teknologi, der anvendes i test, kan inddrages. De kan også omfatte personer, som ikke er involveret i test f.eks. fra forretningsanalytikere, kunder eller klienter
- De kan give tidlig feedback til udviklerne
- De hjælper teamet med at blive bekendt med softwaren, mens den fremstilles
- De er effektive, når driftsfejl analyseres
- De tillader, at forskelligartede teknikker bliver anvendt

Erfaringsbaserede test har følgende ulemper:

- De passer dårligt i systemer, der kræver detaljeret testdokumentation
- Det er svært at opnå et højt niveau af reproducerbarhed
- Man kan ikke vurdere dækningen præcist
- Testene er dårligt egnet til efterfølgende automatisering

Når testerne anvender reaktive og heuristiske tilgange, anvender de normalt erfaringsbaserede tests, der er mere reaktive over for hændelser end de forud planlagte tilgange. Derudover foregår afviklingen og evalueringen samtidig. Nogle strukturerede tilgange til erfaringsbaserede tests er ikke helt dynamiske, dvs. at testene ikke oprettes helt på samme tid, som når testeren afvikler testen. Det kan f.eks. være tilfældet, når fejlgætning anvendes til at målrette visse aspekter til testobjektet, inden testafviklingen.

Bemærk, at selvom der fremlægges nogle idéer til dækningen af de teknikker, som beskrives her, så har erfaringsbaserede testteknikker ingen formelle dækningskriterier.

3.3.1 Fejlgætning

Når man anvender fejlgætningsteknikken, skal testanalytikeren anvende sine erfaringer til at gætte de potentielle fejl, der kan have opstået, da koden blev designet og udviklet. Når de forventede fejl er blevet identificeret, skal testanalytikeren afgøre de bedste metoder til at afsløre de fundne defekter. F.eks. hvis en testanalytiker forventer, at softwaren viser afvigelser, når man indtaster en ugyldig kode, skal man udføre tests,

så der indtastes en række forskellige værdier i kodefeltet for at verificere, om fejlen faktisk fandt sted og ledte til en defekt, der kan opfattes som en afvigelse, når tests afvikles.

Ud over at blive anvendt som en testteknik, kan fejlgætning også være anvendelig i en risikoanalyse til at identificere potentielle afvigelsestilstande. [Myers11]

Anvendelighed
Fejlgætning udføres primært i løbet af integrations- og systemtest, men kan anvendes på hvilket som helst testniveau. Denne teknik anvendes ofte sammen med andre teknikker og hjælper med at udvide de eksisterende testcases omfang. Fejlgætning kan også anvendes effektivt, når man tester releasen af softwaren til at teste almindelige defekter, inden man påbegynder mere grundige tests og scriptede tests.

Begrænsninger/Udfordringer
De følgende begrænsninger og udfordringer er gældende for fejlgætning:

- Dækningen er svær at vurdere, og den varierer i høj grad af testanalytikerens kapacitet og erfaring
- Den kan bedst anvendes af en erfaren tester, der er bekendt med de defekttyper, som generelt introduceres i den type kode, der bliver testet
- Den bliver ofte anvendt, men bliver typisk ikke dokumenteret, så den kan være sværere at reproducere end andre typer tests
- Testcases kan dokumenteres, men på en måde, som kun forfatteren forstår og kan reproducere

Dækning
Når der anvendes en defekt taksonomi, bliver dækningen afgjort ved at tage antallet af taksonomielementer, der testes, og dividere dem med det samlede antal af taksonomielementer, og gengive dækningen i procenter. Uden en taksonomi er dækningen begrænset til testerens erfaring og viden, såvel som den tilgængelige tid. Mængden af de defekter, der bliver fundet vha. denne teknik varierer efter, hvor godt testeren kan spore ind på problematiske områder.

Defekttyper
De typiske defekter er normalvis de, der defineres i den pågældende defekttaksonomi, eller de, som testanalytikeren "gætter", men som ikke blev fundet i black-box testen.

3.3.2 Tjeklistebaseret test

Når erfarne testanalytikere anvender tjeklistebaserede testteknikker, benytter de en højniveau generaliseret liste over elementer, der skal noteres, tjekkes eller huskes, eller et regelsæt eller kriterier, som testelementet skal verificeres i forhold til. Disse tjeklister udarbejdes baseret på et sæt af standarder, erfaringer og andre overvejelser. F.eks., kan en standardtjekliste for brugergrænseflader anvendes som grundlag for at teste et program. I agil softwareudvikling kan tjeklister opbygges ud fra acceptkriterierne for en userstory.

Anvendelighed
Tjeklistebaseret tests er mest effektive i projekter med et erfarent testteam, der har kendskab til den software, der testes, eller har kendskab til det område, der dækkes i tjeklisten (f.eks. for at have succes med at anvende en brugergrænsefladetjekliste, kan en testanalytiker have kendskab til anvendelsen af brugergrænsefladetests, men ikke det specifikke system, der testes). Da tjeklister er på højt niveau og generelt mangler de detaljerede trin, som man ofte finder i testcases og testprocedurer, anvendes testerens viden i stedet til at udfylde hullerne. Ved at fjerne de detaljerede trin, er tjeklisterne nemme at vedligeholde og kan anvendes på flere ensartede releases.

Tjeklister er velegnede til projekter, hvor softwaren releases og gennemgår hurtige ændringer. Denne hjælp kan reducere både forberedelsen og vedligeholdelsestiden for testdokumenterne. De kan anvendes på ethvert testniveau og anvendes også til regressionstest og smoketest.

Begrænsninger/Udfordringer
Tjeklisternes høje niveau kan påvirke testresultaternes evne til at blive reproduceret. Det er muligt, at flere testere kan tolke tjeklisterne forskelligt og følge forskellige tilgange for at udføre punkterne på tjeklisten. Det

kan give forskellige testresultater, selvom det er den samme tjekliste, der anvendes. Dette kan også lede til en bredere dækning, men reproducerbarheden vil i nogle tilfælde blive ofret. Tjeklister kan også lede til for høj tillid til det dækningsniveau, der opnås, da de faktiske tests afhænger af testerens dømmekraft. Tjeklister kan afledes af mere detaljerede testcases eller lister og vokse med tiden. Tjeklister skal vedligeholdes for at sikre, at de løbende dækker de vigtigste aspekter af den software, som testes.

Dækning
Dækning kan afgøres ved at tage antallet af punkter på tjeklisten divideret med det samlede antal af tjeklistepunkter og at skrive dækningen i procenter. Dækningen er lige så god som tjeklisten, men pga. tjeklistens høje niveau, kan resultaterne variere alt efter den testanalytiker, der afvikler punkterne på tjeklisten.

Defekttyper
Typiske defekter, der kan findes med denne teknik, forårsager afvigelser som følge af variation i data, trinenes sekvens eller testens generelle workflow.

3.3.3 Udforskende test

Udforskende tests karakteriseres af, at testeren lærer om testobjektet og dets defekter samtidig med, at testarbejdet planlægges og designes, og at testene afvikles, og resultaterne indrapporteres. Testeren justerer dynamisk testmålene i løbet af afviklingen og forbereder kun en kortfattet dokumentation [Whittaker09].

Anvendelighed
Gode udforskende tests planlægges og er interaktive og kreative. De kræver kun lidt dokumentation om systemet for at blive testet og anvendes ofte i situationer, hvor dokumenterne ikke er tilgængelige eller tilstrækkelige til andre testteknikker. Udforskende tests anvendes ofte til at bidrage til andre testteknikker og som grundlag for udviklingen af yderligere testcases. Udforskende tests bliver ofte anvendt i agil softwareudvikling for at udføre userstories på fleksibel og hurtig vis med et minimalt niveau af dokumentation. Teknikken kan også anvendes i projekter, der bruger en sekventiel udviklingsmodel.

Begrænsninger/Udfordringer
Dækningen i udforskende tests kan være sporadisk, og det kan være svært at reproducere de udførte tests. De teknikker, man anvender til at håndtere udforskende tests, er testcharters til at tildele de områder, der skal dækkes i en testsession, og time-boxing til at afgøre den tilladte mængde testtid. I slutningen af en testsession kan testmanageren afholde en debriefing-session for at indsamle testenes resultater og afgøre testcharters for de næste testsessioner.

En anden udfordring ved udforskende testsessioner er at spore dem nøjagtigt i et teststyringssystem. Det kan nogle gange gøres ved at oprette testcases, der faktisk er udforskende testsessioner. Det giver tid til, at de eksplorative tests og den planlagte dækning kan spores med andre testmetoder.

Da det kan være svært at reproducere med udforskende tests, kan det også udgøre et problem, når man skal huske trinene til at genskabe en afvigelse. Nogle organisationer anvender optage / afspille funktionerne i testautomatiseringsværktøjet for at optage de trin, som den udforskende tester foretager. Derved opnår man de samlede fortegnelser over alle aktiviteter i løbet af den udforskende testsession (eller anden erfaringsbaseret testsession). Det kan forekomme ensformigt at analysere detaljerne for at finde den faktiske årsag til afvigelserne, men som minimum er der optagelser af alle de involverede trin.

Der kan anvendes andre værktøjer til at optage de udforskende testsessioner, men de optager ikke de forventede resultater, da de ikke optager GUI-interaktion. I dette tilfælde skal de forventede resultater skrives ned, så man kan foretage en ordentlig analyse af afvigelserne om nødvendigt. Generelt anbefales det, at de noter, der bliver skrevet ned, når man udfører udforskende tests, understøtter reproducerbarheden der, hvor det er påkrævet.

Dækning
Testcharters kan være designet til særlige opgaver, målsætninger og enheder, der kan leveres. Udforskende testsessioner planlægges til at imødekomme disse kriterier. Charteret kan også identificere, hvor man skal fokusere sin indsats i testen, hvad der er op og ned i forhold til testsessionens omfang, og hvilke resurser,

der bør tilegnes færdigafviklingen af de planlagte tests. En testsession kan bruges til at fokusere på visse defekttyper og andre mulige problematiske områder, der kan tilses uden den formelle tilgang fra manuskriptbaseret tests.

Defekttyper
De typer af defekter, der bliver fundet i udforskende tests, er scenariebaserede problemer, der blev overset i løbet af de scriptbaserede og funktionelle egnethedstests, såvel som problemer, der ligger mellem funktionelle grænser og workflows-relaterede problemer. Performance og sikkerhedsproblemer bliver også nogle gange afdækket i løbet af de udforskende tests.

3.3.4 Defektbaseret testteknikker

En defektbaseret testteknik er en testteknik, hvor den defekttype, man søger efter, anvendes som grundlag for testdesignet, og hvor testene udledes systematisk fra det, som er kendt om defekttypen. I modsætning til black-box tests, der udleder sine tests fra testgrundlaget, udleder defektbaserede tests ud fra lister, der fokuserer på defekter. Generelt kan listerne være organiseret i defekttyper, rodårsager, afvigelsessymptomer og anden defektrelateret data. Standardlister kan anvendes til flere softwaretyper og er ikke produktspecifikke. Vha. disse lister kan man følge industristandarder til at udlede bestemte tests. Ved at følge industrispecifikke lister, kan man måle tilfælde af defekter på tværs af projekter og endda på tværs af organisationer. De mest almindelige defektlister er de, der er tilknyttet en organisation eller et projekt, og som anvender særlig ekspertise og erfaringer.

Defektbaserede tests kan også anvende lister med identificerede risici og risikoscenarier som grundlag for målrettede tests. Denne type testteknik lader en testanalytiker spore sig ind på specifikke defekttyper eller at arbejde sig systematisk igennem en liste over kendte og almindelige defekter af en særlig type. Ud fra disse oplysninger opretter testanalytikeren testbetingelser og testcases, der kan forårsage, at defekter opstår, hvis betingelserne finder sted.

Anvendelighed
Defektbaserede tests kan anvendes på ethvert testniveau, men bliver typisk anvendt ved systemtests.

Begrænsninger/Udfordringer
Der findes flere defekttaksonomier, og de kan fokuseres på en særlig type test f.eks. brugervenlighed. Det er vigtigt, at man vælger en taksonomi, der er anvendelig på den software, man tester, hvis sådan en findes. F.eks. kan det være tilfældet, at der ikke findes nogen tilgængelige taksonomier for innovativ software. Nogle organisationer har samlet deres egne taksonomier af sandsynlige eller ofte sete defekter. Uanset hvilken defekttaksonomi, der anvendes, er det vigtigt at definere den forventede dækning, inden man påbegynder testen.

Dækning
Teknikken giver dækningskriterierne, der anvendes til at afgøre, hvornår alle de anvendelige testcases er blevet identificeret. Dækningselementer kan være strukturelle elementer, specifikationselementer, brugerscenarier eller en blanding af disse afhængigt af defektlisten. I forhold til praktiske hensyn vil dækningskriterierne fra defektbaserede testteknikker ofte være mindre systematiske end black-box testteknikker, da den eneste generelle regel for dækning er givet på forhånd, og de specifikke beslutninger om, hvad der udgør grænsen for anvendelig dækning, er et spørgsmål om skøn. Ligesom med andre teknikker betyder dækningskriterierne ikke, at hele sættet af tests er fuldendt, men det kan i stedet betyde, at de foreslåede defekter ikke længere peger på nogen brugbare tests baseret på den teknik.

Defekttyper
De defekttyper, der opdages, afhænger typisk af den anvendte defekttaksonomi. F.eks. hvis en defektliste for brugergrænseflader bliver anvendt, vil hovedparten af de fundne defekter antageligvis være relateret til brugergrænseflader, men andre defekter kan anvendes som et biprodukt af den specifikke test.

3.4 Anvendelse af de mest passende teknikker

Black-box og erfaringsbaserede testteknikker er mest effektive, når de bliver anvendt sammen. Erfaringsbaserede testteknikker udfylder hullerne i dækningen, der er resultatet af systematiske svagheder i black-box testteknikker.

Der findes ikke en perfekt teknik, der passer til alle situationer. Det er vigtigt for testanalytikeren at forstå hver tekniks fordele og ulemper og at være i stand til at vælge den bedste teknik eller det bedste sæt teknikker til den pågældende situation i forhold til projekttypen, tidsplanen, adgang til oplysninger, testerens evner og andre faktorer, der kan påvirke valget.

I beskrivelsen af black-box og erfaringsbaserede testteknikker (se henholdsvis afsnit 3.2 og 3.3) vejleder de oplysninger, der står skrevet om "anvendelighed", "begrænsninger/udfordringer" og "dækning" testanalytikeren i valget af hvilken testteknik, der skal anvendes.

4. Test af softwarens kvalitetsegenskaber 180 minutter

Nøgleord

beskyttelse mod brugerfejl, brugergrænsefladens æstetik, brugeroplevelse, brugervenlighed, brugsegnethed, funktionel egnethed, funktionel fuldstændighed, funktionel hensigtsmæssighed, funktionel korrekthed, kompabilitet, læringsegnethed, Software Usability Measurement Inventory (SUMI), tilgængelighed, tværoperationalitet, Website Analysis and MeasureMent Inventory (WAMMI)

Læringsmål for test af softwarens kvalitetsegenskaber

4.1 Introduktion

Ingen læringsmål

4.2 Kvalitetsegenskaber for test af forretningsdomæne

TA-4.2.1 (K2) Forklar, hvilke testteknikker, der passer til at teste den funktionelle fuldstændighed, korrekthed og hensigtsmæssighed

TA-4.2.2 (K2) Definer de typiske defekter, der er mål for test af den funktionelle fuldstændighed, korrekthed og hensigtsmæssighed

TA-4.2.3 (K2) Definer, hvornår den funktionelle fuldstændighed, korrekthed og hensigtsmæssighed skal testes i softwareudviklingens livscyklus

TA-4.2.4 (K2) Forklar de tilgange, der kan anvendes til at verificere og validere både implementeringen af brugervenlighedskravene og opfyldelsen af brugerens forventninger

TA-4.2.5 (K2) Forklar testanalytikerens rolle i tværoperationalitetstests herunder identificering af de defekter, den retter sig mod

TA-4.2.6 (K2) Forklar testanalytikerens rolle i flytbarhedstests herunder identificering af de defekter, den retter sig mod

TA-4.2.7 (K4) Inden for omfanget af testanalytikerens rolle at bestem de testbetingelser, der er påkrævet for at verificere de funktionelle og/eller ikke-funktionelle kvalitetsegenskaber for et givet sæt af krav

4.1 Introduktion

Det forrige kapitel beskrev de specifikke testteknikker, som er tilgængelige for testeren. Dette kapitel diskuterer anvendelsen af teknikkerne til at evaluere de egenskaber, der anvendes til at beskrive applikationernes eller systemernes kvalitet.

Denne syllabus beskriver de kvalitetsegenskaber, som skal evalueres af testanalytikeren. De egenskaber, der evalueres af den tekniske testanalytiker diskuteres i Advanced Technical Test Analyst syllabus [CTAL-TTA].

Den beskrivelse af produktets kvalitetsegenskaber, som står skrevet i ISO 25010 [ISO25010], anvendes som en vejledning til at beskrive egenskaberne. ISO-softwarens kvalitetsmodel opdeler produktkvaliteten i forskellige egenskaber for produktkvaliteten, hvor hver af dem kan have underegenskaber. De kan ses i tabellen nedenfor sammen med en indikation af, hvilke egenskaber/underegenskaber, der dækkes af syllabi for testanalytikeren og den tekniske testanalytiker:

Egenskaber	Underegenskaber	Testanalytiker	Teknisk testanalytiker
Funktionel egnethed	Funktionel korrekthed, funktionel hensigtsmæssighed, funktionel fuldstændighed	X	
Pålidelighed	Modenhed, fejltolerance, genopretningsevne, tilgængelighed		X
Brugervenlighed	Anerkendelse af passende genkendelighed, læringsegnethed, brugsegnethed, brugergrænsefladens æstetik, beskyttelse mod brugerfejl, tilgængelighed	X	
Ydelsens effektivitet	Adfærd over tid, udnyttelse af resurser, kapacitet		X
Vedligeholdelsesegnethed	Analyserbarhed, modificerbarhed, testbarhed, modulerbarhed, genanvendelighed		X
Flytbarhed	Tilpasningsegnethed, installerbarhed, udskiftningsegnethed	X	X
Datasikkerhed	Fortrolighed, integritet, uafviselighed, ansvarlighed, autenticitet		X
Kompatibilitet	Sameksistens		X
	Tværoperationalitet	X	

Selvom denne fordeling af arbejdsopgaver kan variere i forskellige organisationer, er det denne, som følges i de tilknyttede ISTQB® syllabi.

For alle kvalitetsegenskaber og underegenskaber , der beskrives i dette afsnit, skal de typiske risici anerkendes, så en passende teststrategi kan udformes og dokumenteres. Test af kvalitetsegenskaber kræver særlig opmærksomhed i forhold til SDLC-timing, de påkrævede værktøjer, software, tilgængelig dokumentation og teknisk ekspertise. Uden en strategi til at håndtere hver egenskab og dens unikke testbehov, kan det ske, at testeren ikke har tilstrækkelig tid til planlægning, opmanding og afvikling ifølge planen [Bath14]. Dele af disse tests f.eks. brugervenlighedstest kan kræve tildeling af særlig arbejdskraft, omfattende planlægning, dedikerede laboratorier, specifikke værktøjer, specialiserede testfærdigheder og - i de fleste tilfælde - en betydelig mængde tid. I nogle tilfælde kan brugervenlighedstests udføres af en separat gruppe af eksperter i brugervenlighed eller brugeroplevelser.

Selvom testanalytikeren ikke er ansvarlig for kvalitetskarakteristika, der kræver en mere teknisk tilgang, er det vigtigt, at testanalytikeren er bevidst om andre karakteristikker og forstår de overlappende testområder. F.eks. vil et testobjekt, der fejler performancetesten højst sandsynligt også fejle brugervenlighedstesten, hvis

den er for langsom til, at brugeren kan anvende den korrekt. På samme måde er et testobjekt med tværoperationalitetsproblemer med visse komponenter formodentlig heller ikke klar til flytbarhedstest, da det vil overskygge de mere basale problemer, når miljøet ændres.

4.2 Kvalitetsegenskaber for test af forretningsdomæne

Test af funktionel egnethed er det primære fokus for en testanalytiker. Test af funktionel egnethed fokuserer på hvad testobjektet gør. Testgrundlaget for test af funktionel egnethed er generelt krav, specifikationer, specifik domæneekspertise eller implicitte behov. Funktionel egnethedstests varierer i forhold til testniveauet, hvori de udføres, og kan også påvirkes af SDLC'en. F.eks. vil en funktionel egnethedstest, der udføres i løbet af integrationstesten, teste funktionaliteten af samspillet mellem komponenter, der implementerer en enkelt, specificeret funktion. På systemtestniveau omfatter de funktionelle egnethedstests test af systemets funktionalitet som helhed. For systemer af systemer vil funktionel egnethedstests primært fokusere på end-to-end tests på tværs af de integrerede systemer. Man anvender et bredt udvalg af testteknikker i løbet af en test af funktionel egnethed (se kapitel 3).

I agil softwareudvikling omfatter test af funktionel egnethed generelt de følgende:

- Test af den specifikke funktionalitet (f.eks. en userstory), der er planlagt til implementering i den specifikke iteration
- Regressionstest for al uændret funktionalitet

Ud over den test af funktionel egnethed, der er dækket i dette afsnit, er der også visse kvalitetsegenskaber, der er en del af testanalytikerens ansvarsområde, som regnes for ikke-funktionelle (fokuser på "hvordan" testobjektet leverer funktionaliteten) testområder.

4.2.1 Test af funktionel korrekthed

Funktionel korrekthed omfatter verifikation af, om applikationen overholder de specificerede eller underforståede krav og kan også omfatte nøjagtighed i beregninger. Test af funktionel korrekthed anvender mange af de testteknikker, der beskrives i kapitel 3, og de benytter ofte specifikation eller et legacy system som testorakel. Test af funktionel korrekthed kan udføres på ethvert testniveau og retter sig mod ukorrekt håndtering af data eller situationer.

4.2.2 Test af funktionel hensigtsmæssighed

Funktionel hensigtsmæssighed indebærer evaluering og validering af et sæt af funktioners hensigtsmæssighed i forhold til de specifikke, tiltænkte opgaver. Denne form for test kan baseres på det funktionelle design (f.eks. usecases og/eller userstories). Test af funktionel hensigtsmæssighed udføres generelt i løbet af systemtesten, men kan også udføres i løbet af de senere stadier af integrationstesten. De defekter, der opdages i denne test, er en indikation på, at systemet ikke er i stand til at imødekomme brugerens behov på acceptabel vis.

4.2.3 Test af funktionel fuldstændighed

Test af funktionel fuldstændighed udføres for at afgøre dækningsgraden af den specifikke opgave og brugerens målsætninger i den implementerede funktionalitet. Det er vigtigt med sporbarhed mellem specifikationselementer (f.eks. krav, userstories, usecases) og den implementerede funktionalitet (f.eks. funktion, komponent, workflow) for at gøre det muligt at afgøre den påkrævede funktionel fuldstændighed. Måling af den funktionelle fuldstændighed kan variere i forhold til det specifikke testniveau og/eller den anvendte SDLC. F.eks. kan den funktionelle fuldstændighed for agil softwareudvikling være baseret på implementering af userstories og egenskaber. Funktionel fuldstændighed for systemintegrationstests kan fokusere på dækningen af højniveau forretningsprocesser.

Der findes teststyringsværktøjer, der understøtter test af fuldstændighed, hvis testanalytikeren opretholder sporbarhed mellem testcases og de funktionelle specifikationselementer. Et lavt niveau af funktionel fuldstændighed er en indikation på, at systemet ikke er implementeret fuldstændigt.

4.2.4 Tværoperationalitetstests

Tværoperationalitetstests verificerer udvekslingen af oplysninger mellem to eller flere systemer eller komponenter. Sådanne tests fokuserer på evnen til at udveksle oplysninger og derefter at anvende de oplysninger, der er blevet udvekslet. Testen kan dække de tiltænkte målrettede miljøer (herunder de variationer i hardware, software, middleware, operativsystem osv.) for at sikre, at dataudvekslingen fungerer korrekt. I virkeligheden er det kun muligt at gennemføre testen for et relativt lille antal miljøer. Derfor kan tværoperationalitetstests være begrænset til en udvalgt repræsentativ gruppe af miljøer. At specificere tests til tværoperationalitet kræver, at kombinationen af tiltænkte målmiljøer identificeres, konfigureres og er tilgængelige for testteamet. Disse miljøer skal så testes vha. af et udvalg af passende, funktionelle testcases, der påvirker de forskellige dataudvekslingspunkter i miljøet.

Tværoperationalitet relaterer til, hvordan forskellige komponenter og softwaresystemer interagerer med hinanden. Software med gode tværoperationelle egenskaber kan integreres med et antal andre systemer uden at kræve større ændringer eller betydelig påvirkning på ikke-funktionel adfærd. Antallet af ændringer og den indsats, der er påkrævet for at implementere og teste dem, kan anvendes som et omfang af tværoperationalitet.

Test af softwarens tværoperationalitet kan f.eks. fokusere på følgende designegenskaber:

- Brug af kommunikationsstandarder, der anvendes inden for hele branchen, f.eks. XML
- Evnen til automatisk at opfange den kommunikation, der er nødvendig for de systemer, softwaren interagerer med

Tværoperationalitetstest kan have særlig betydning for de følgende:

- Softwareprodukter og værktøjer, der er kommerciel standardsoftware
- Applikationer baseret på et system af systemer
- Systemer baseret på Internet of Things
- Web services med forbindelse til andre systemer

Denne type tests udføres i løbet af komponentintegration- og systemintegrationtests. På systemintegrationsniveau udføres denne type tests for at afgøre, hvor godt det fuldt udviklede system interagerer med andre systemer. Da systemer kan samarbejde på flere niveauer, skal testanalytikeren forstå disse interaktioner og være i stand til at skabe de betingelser, der kan udøve de forskellige interaktioner. Hvis to systemer f.eks. udveksler data, skal testanalytikeren være i stand til at skabe de nødvendige data og de transaktioner, der kræves, for at udføre dataudvekslingen. Det er vigtigt at huske, at ikke alle interaktioner nødvendigvis er tydeligt beskrevet i kravdokumenterne. I stedet bliver mange af disse interaktioner kun defineret i systemarkitekturen og designdokumenterne. Testanalytikeren skal være i stand til og forberedt på at undersøge disse punkter med informationsudvekslinger mellem systemer og mellem systemerne og deres miljøer for at sikre, at de alle bliver testet. Teknikker som ækvivalenspartitionering, grænseværdianalyse, beslutningstabeller, tilstandsovergangsdiagrammer, usecases og parvis test kan alle anvendes i tværoperationalitetstests. De typiske defekter er blandt andet ukorrekt dataudveksling mellem interagerende komponenter.

4.2.5 Evaluering af brugervenlighed

Testanalytikere befinder sig ofte i en position, hvor de skal koordinere og understøtte evalueringen af brugervenlighed. Dette kan omfatte specificering af brugervenlighedstests eller at testeren er en moderator, der arbejder sammen med brugerne om testen. For at kunne gøre det effektivt, skal testanalytikeren forstå de vigtigste aspekter, mål og tilgange, der indgår i disse typer af tests. Se venligst ISTQB® Specialist syllabus for brugervenlighedstests [ISTQB_UT_SYL_SYL] for at se flere detaljer.
Det er vigtigt at forstå, hvorfor brugerne kan have svært ved at anvende systemet eller ikke har positive brugeroplevelser (UX) (f.eks. for software til underholdning). Derfor er det nødvendigt at forstå, at ”bruger” kan beskrive en lang række af forskellige typer personligheder fra IT-eksperter til individer med handicap.

4.2.5.1 Brugervenlighedsaspekter

De følgende tre aspekter betragtes i dette afsnit:

- Brugervenlighed
- Brugeroplevelsen (UX)
- Tilgængelighed

Brugervenlighed
Brugervenlighedstests bruges til softwaredefekter, der påvirker brugerens evne til at udføre opgaver via brugergrænsefladen. Sådanne defekter kan påvirke brugerens evne til at nå sine mål på effektiv, virkningsfuld eller tilfredsstillende vis. Brugervenlighedsproblemer kan lede til forvirring, fejl, forsinkelser eller direkte afvigelse, når brugeren forsøger at udføre bestemte opgaver.

De følgende er brugervenlighedens underegenskaber [ISO 25010], se deres definitioner i [ISTQB_GLOSSARY]:

- Passende genkendelighed (f.eks. forståelighed)
- Læringsegnethed
- Brugsegnethed
- Brugergrænsefladens æstetik (f.eks. udseende)
- Grad af beskyttelse mod brugerfejl
- Tilgængelighed (se nedenfor)

Brugeroplevelse (UX)
Brugeroplevelsesevaluering ser nærmere på hele brugeroplevelsen med testobjektet, ikke blot den direkte interaktion. Det er særligt vigtigt for testobjekter, hvor faktorer som nydelse og brugertilfredshed er vigtig for forretningens succes.

Typiske faktorer, der påvirker brugeroplevelsen, omfatter de følgende:

- Brandets image (f.eks. brugerens tillid til fabrikanten)
- Interaktiv adfærd
- Testobjektets hjælpsomhed omfatter hjælpesystemer, support og træning

Tilgængelighed
Det er vigtigt at betragte softwarens tilgængelighed for brugere med særlige behov eller begrænsninger. Det omfatter også de, der har handicap. Tilgængelighedstests bør følge de relevante standarder som Web Content Accessibility Guidelines (WCAG) og lovgivning som Disability Discrimination Acts (Nordirland, Australien), Equality Act 2010 (England, Skotland, Wales) og afsnit 508 (USA). Tilgængeligheden skal ligesom brugervenligheden overvejes, når man gennemfører designet. Testen finder ofte sted i løbet af integrationsniveauerne og fortsætter i løbet af systemtesten og ind i accepttestniveauerne. Defekter bliver typisk fundet, når softwaren ikke formår at leve op til de fastlagte regler eller standarder, der er beskrevet for softwaren.

Typiske midler til at forbedre tilgængeligheden fokuserer på de muligheder, der leveres, for at brugere med handicap kan interagere med applikationen. Disse omfatter:

- Stemmegenkendelse til input
- Sikring af at indhold, der ikke er tekst, og som vises for brugeren, også har en alternativ tekst
- Mulighed for at ændre størrelse på teksten uden tab af indhold eller funktionalitet

Tilgængelighedsvejledninger understøtter testanalytikeren ved at levere en kilde med oplysninger og tjeklister, der kan anvendes til testen (eksempler på en tilgængelighedsvejledning kan ses i [ISTQB_UT_SYL]). Derudover er værktøjer og browserplugins tilgængelige for at hjælpe testeren med at identificere tilgængelighedsproblemer som f.eks. farver på hjemmesiderne, der ikke følger vejledningen for farveblindhed.

4.2.5.2 Tilgange til brugervenlighedsevaluering

Brugervenlighed, brugeroplevelse og tilgængelighed kan testes via en eller flere af de følgende tilgange:

- Brugervenlighedstest
- Brugervenlighedsreviews
- Brugervenlighedsspørgeundersøgelser og -spørgeskemaer

Brugervenlighedstest
Brugervenlighedstests evaluerer, hvor let brugerne kan anvende eller lære at anvende systemet til at opnå et specifikt mål inden for en særlig kontekst. Brugervenlighedstests forsøger at måle:

- Egnethed - testobjektets evne til at lade brugerne opnå de specifikke mål med nøjagtighed og fuldstændighed i en specifik brugssammenhæng
- Effektivitet - testobjektets evne til at lade brugerne anvende en passende mængde resurser i forhold til den opnåede effektivitet i en specifik brugssammenhæng
- Tilfredshed - testobjektets evne til at tilfredsstille brugere i en specifik brugssammenhæng

Det er vigtigt at bemærke, at det at designe og specificere brugervenlighedstests ofte udføres af testanalytikeren sammen med testere, der har særlige færdigheder inden for brugervenlighedstests og brugervenlighedsdesignteknikere, der forstår sig på en menneskeorienteret designproces (se [ISTQB_UT_SYL] for yderligere detaljer).

Brugervenlighedsreviews
Inspektioner og reviews er en slags tests, der udføres fra et brugervenlighedsperspektiv. De skal hjælpe til at øge brugerens grad af involvering. Det kan være omkostningseffektivt at finde brugervenlighedsproblemer i kravsspecifikationer og design tidligt i SDLC. Heuristisk evaluering (systematisk inspektion af en brugergrænseflades design med henblik på brugervenlighed) kan anvendes til at finde brugervenlighedsproblemer i designet, så de kan adresseres i den iterative designproces. Det indebærer at have en lille gruppe evaluatorer til at undersøge brugergrænsefladen eller vurdere efterlevelse af de anerkendte brugervenlighedsprincipper (”heuristikker ”). Review er mest effektiv, når brugergrænsefladen er synlig. F.eks. er eksempler på skærmbilleder nemmere at forstå og tolke, end hvis man blot at beskriver kravene til funktionaliteten på et specifikt skærmbillede. Visualisering er vigtig for et tilstrækkeligt brugervenlighedsreview af dokumentet.

Brugervenlighedsundersøgelser og -spørgeskemaer
Undersøgelser og spørgeskemateknikker kan anvendes til at indhente observationer og feedback om brugernes adfærd med systemet. Standardiseret og offentligt tilgængelige undersøgelser som SUMI (Software Usability Measurement Inventory) og WAMMI (Website Analysis and MeasureMent Inventory) gør det muligt at benchmarke ud fra en database med tidligere brugervenlighedsmålinger. Da SUMI også giver håndgribelige målinger af brugervenlighed, giver det et sæt af færdiggørelses-/acceptkriterier.

4.2.6 Flytbarhedstest

Flytbarhedstests forholder sig til den grad, som en softwarekomponent eller et system kan overføres til sit tiltænkte miljø, enten som en ny installation eller fra et eksisterende miljø.

ISO 25010-klassifikationen af produktegenskaber omfatter de følgende underegenskaber af flytbarhed:

- Installerbarhed
- Tilpasningsevne
- Udskiftningsegnethed

Opgaven med at identificere risici og at designe tests til flytbarhedsegenskaber er delt mellem testanalytikeren og den tekniske testanalytiker (se [ISTQB_ALTTA_SYL] afsnit 4.7).

4.2.6.1 Test af installerbarhed

Test af installerbarhed udføres på softwaren og de skrevne procedurer, der anvendes til at installere og afinstallere softwaren i sit målmiljø.

De typiske testmålsætninger, som testanalytikeren fokuserer på, omfatter:

- At sikre, at de forskellige konfigurationer af softwaren kan installeres. Hvis der er et stort antal parametre der skal konfigureres, kan testanalytikeren designe tests, der anvender parvis testteknik til at reducere antallet af parameterkombinationer, som skal testes, og til at fokusere på særligt interessante konfigurationer (f.eks. de, der oftest anvendes)
- Test af installations- og afinstallationsprocedurernes funktionelle korrekthed
- Afvikle passende funktionelle tests efter en installation eller afinstallation for at spore mulige defekter, der kan være introduceret (f.eks. ukorrekte konfigurationer, utilgængelige funktioner)
- Identificere brugervenlighedsproblemer i installations- og afinstallationsprocedurer (f.eks. til at godkende, at brugerne har adgang til forståelige instruktioner og feedback/fejlmeddelelser, når proceduren udføres)

4.2.6.2 Test af tilpasningsevne

Test af tilpasningsevne undersøger, om en given applikation kan tilpasses til effektivt at fungere korrekt i alle de tiltænkte målmiljøer (hardware, software, middleware, operativsystem, cloud, osv.). Testanalytikeren understøtter testen af tilpasningsevne ved at identificerer kombinationerne af de tiltænkte målmiljøer (f.eks. versioner af forskellige understøttede operativsystemer til mobile enheder og forskellige versioner af de browsere, der kan anvendes), og ved at designe tests der dækker kombinationer af disse miljøer. Disse miljøer skal så testes ved at anvende et udvalg af funktionel egnede testcases, der kan afvikle de forskellige komponenter, som er tilgængelige i miljøet.

4.2.6.3 Test af udskiftningsegnethed

Test af udskiftelighed fokuserer på softwarekomponenterne eller et systems forskellige versioners evne til at blive udskiftet med en anden. Det kan være særligt relevant for systemarkitekturer baseret på Internet of Things, hvor der kan ske udskiftning af forskellige hardwareenheder og/eller softwareinstallationer. F.eks. kan enhardwareenhed, der anvendes på et lager til at registrere og føre kontrol med lagerbestanden, erstattes af en mere avanceret hardwareenhed (f.eks. med en bedre scanner), eller der kan ske opgradering af den installerede software til en ny version, der gør det muligt automatisk at afsende ordrer om lageropfyldning til leverandørens system.

Tests af udskiftelighed kan udføres af testanalytikeren parallelt med den funktionelle integrationstests, hvor mere end én alternativ komponent er tilgængeligt til integration i det samlede system.

5. Review 120 minutter

Nøgleord

tjeklistebaseret review

Læringsmål for reviews

5.1 Introduktion

Ingen læringsmål

5.2 Anvendelse af tjeklister i reviews

TA-5.2.1 (K3) Identificer problemer i en kravspecifikation i henhold til de oplysninger om tjeklister, der står i denne syllabus

TA-5.2.2 (K3) Identificer problemer i en userstory i henhold til de oplysninger om tjeklister, der står i denne syllabus

5.1 Introduktion

Testanalytikerne skal være aktive deltagere i reviewprocessen og levere deres egne, unikke synspunkter. Når de udføres korrekt, kan reviews udgøre det væsentligste og mest omkostningseffektive bidrag til den overordnede leverede kvalitet.

5.2 Anvendelse af tjeklister i reviews

Når testgrundlaget reviewes er den mest almindelige teknik, der anvendes af testanalytikeren, Tjeklistebaserede reviews. Tjeklister anvendes i løbet af reviewet for at få deltagerne til at tjekke særlige punkter i løbet af reviewet. De kan også hjælpe med at gøre reviewet mere upersonligt (f.eks. ”Det er den samme tjekliste vi bruger ved hvert review. Det er ikke kun dit arbejdsprodukt vi tilser sådan.”).

Tjeklistebaserede reviews kan udføres generisk i alle reviews eller kan fokusere på specifikke kvalitetsegenskaber, områder eller typer af dokumenter. F.eks. kan en generisk tjekliste verificere de generelle dokumentegenskaber f.eks. unikke identifikatorer, at ingen henvisninger er markeret som ”endnu uafklaret”, passende formatering og lignende. En specifik tjekliste for et kravdokument kan indeholde tjek både for den korrekte anvendelse af termerne ”skal” og ”bør”, tjek af testbarhed for hvert skrevet krav osv.

Kravenes format kan også indikere typen af den tjekliste, der skal anvendes. Et kravdokument, der står i et fortællende tekstformat, kan have andre reviewkriterier end de, der er baseret på diagrammer.

Tjeklister kan også være orienteret imod et særligt aspekt som:

- En programmørs/arkitekts færdigheder eller en testers færdigheder - i tilfældet af en testanalytiker, vil en tjekliste for en testers færdigheder være mest passende
- Et særligt risikoniveau (f.eks. sikkerhedskritiske systemer) - tjeklisten vil typisk indeholde de specifikke oplysninger, der behøves på risikoniveauet
- En specifik testteknik - tjeklisten fokuserer på de oplysninger, der er behov for i henhold til særlige teknikker (f.eks. de regler, der skal repræsenteres i en beslutningstabel)
- Et særligt specifikationselement, f.eks. et krav, en usecase eller userstory - disse beskrives i de følgende afsnit og har generelt et andet fokus end dem, der anvendes af en teknisk testanalytiker til review af kode eller arkitektur

5.2.1 Review af krav

De følgende elementer er eksempler på, hvad en kravsorienteret tjekliste kan indeholde:

- Kravets kilde (f.eks. en person, afdeling)
- Hvert kravs testbarhed
- Prioritet af hvert krav
- Acceptkriterier for hvert krav
- Tilgængeligheden af en usecase kaldestruktur, hvis den er passende
- Unik identifikation af hvert krav/usecase/userstory
- Versionering af hvert krav/usecase/userstory
- Sporbarheden til hvert krav fra forretningen/marketingskrav
- Sporbarheden mellem krav og/eller usecases (hvis den er passende)
- Anvendelse af konsekvent terminologi (f.eks. anvendelse af en ordliste)

Det er vigtigt at huske, at hvis et krav ikke er testbart, dvs. at hvis et krav er defineret på en sådan måde, at testanalytikeren ikke kan afgøre, hvordan det skal testes, så findes der en defekt i det krav. F.eks. er et krav, der siger ”Softwaren skal være meget brugervenlig”, ikke testbar. Hvordan kan testanalytikeren afgøre, om softwaren er brugervenlig eller endda meget brugervenlig? Hvis kravet i stedet siger ”Softwaren skal efterleve de brugervenlighedsstandarder, der står i dokument version xxx”, og hvis brugervenlighedsstandarddokumentet findes, så er det et testbart krav. Det er også et overordnet krav, hvis dette ene krav gælder for hvert element i brugergrænsefladen. I dette tilfælde kan dette ene krav afføde mange testcases i en ikke-triviel

applikation. Sporbarheden fra dette krav, eller muligvis fra dokumentet med brugervenlighedsstandarder for testcases, er også nødvendig, for hvis den henviste brugervenlighedsspecikation ændres, så er det nødvendigt, at alle testcases reviews og opdateres efter behov.

Et krav er heller ikke testbart, hvis testeren er ude af stand til at afgøre, hvorvidt testen er bestået eller fejlet, eller at denne er ude af stand til at lave en test, der kan bestås eller fejles. F.eks. er "Systemet skal være tilgængeligt 100% af tiden, 24 timer hver dag, 7 dage hver uge, 365 (eller 366) dage på et år" ikke testbart.

En simpel tjekliste for usecasereviews kan indeholde de følgende spørgsmål:

- Er den primære adfærd (stien) tydeligt defineret?
- Er alle alternative typer adfærd (stier) identificeret samt fejlhåndteret?
- Er alle brugergrænseflademeddelser defineret?
- Er der kun én slags primær adfærd (det bør der være, ellers er der flere usecases)?
- Er alle typer af adfærd testbare?

5.2.2 Review af userstories

I agil softwareudvikling vil kravene som regel bestå af userstories. Disse historier repræsenterer små enheder af påviselig funktionalitet. Hvor en usecase er en brugertransaktion, der bevæger sig igennem flere funktionalitetsområder, er en userstory en mere isoleret egenskab, der generelt befinder sig inden for det tidsmæssige omfang, der kræves for at udvikle den. En tjekliste[1] for en userstory kan indeholde følgende:

- Er den pågældende userstory passende i forhold til målet for iterationen/sprintet?
- Er den pågældende userstory skrevet ud fra den anmodende persons perspektiv?
- Er acceptkriterierne definerede og testbare?
- Er funktioner klart defineret og tydelig?
- Er den pågældende userstory uafhængig af de andre?
- Er den pågældende userstory prioriteret?
- Følger den pågældende userstory det alment benyttede format?
  Som < type bruger > ønsker jeg < et mål >, så jeg < en årsag > [Cohn04]

Hvis den pågældende userstory definerer en ny brugergrænseflade, så kan det, at der anvendes en generisk userstory tjekliste (som den ovennævnte) og en detaljeret tjekliste for brugergrænsefladen, være passende.

5.2.3 Tilpas tjeklisterne

En tjekliste kan tilpasses baseret på følgende:

- Organisation (f.eks. Hvis man ser på forretningspolitik, standarder, konventioner, lovmæssige krav)
- Projekt / udviklingsindsats (f.eks. fokus, tekniske standarder, risici)
- Den type af arbejdsprodukt, der skal reviewes (f.eks. kan review af kode tilpasses til specifikke programmeringssprog)
- Risikoniveauet for, at arbejdsproduktet bliver reviewet
- Testteknikker, der skal anvendes

Gode tjeklister finder problemer og hjælper også med at påbegynde diskussioner vedrørende andre elementer, der måske ikke henvises til direkte i tjeklisten. At anvende en kombination af tjeklister er en god måde at sikre, at reviewet leverer et arbejdsprodukt af højeste kvalitet. Ved at anvende tjeklistebaserede reviews med standardtjeklister som de, der er henvist til i Foundation Level syllabus og udvikling af specifikke organisatoriske tjeklister som de ovenfor viste kan det hjælpe testanalytikeren til at udføre reviewet på effektiv vis.

For yderligere oplysninger om reviews og inspektioner, se da [Gilb93] og [Wiegers03]. Yderligere eksempler på tjeklister kan hentes fra henvisningen i afsnit 7 – Andre kilder.

[1] *Et Eksamensspørgsmål vil give en delmængde af en tjeklisten til en usecases, som du kan besvare spørgsmålet ud fra*

6. Testværktøjer og automatisering 90 minutter

Nøgleord

forberedelse af testdata, nøgleordsdrevet test, testafvikling, testscript

Læringsmål for testværktøjer og automatisering

6.1 Introduktion

Ingen læringsmål

6.2 Nøgleordsdrevet automatisering

TA-6.2.1 (K3) Find frem til de passende aktiviteter til en testanalytiker i et nøgleordsdrevet testprojekt

6.3 Typer af testværktøjer

TA-6.3.1 (K2) Forklar anvendelsen og typer af testværktøjer, der anvendes i testdesign, forberedelse af testdata og testafviklingen

6.1 Introduktion

Testværktøjer kan øge effektiviteten og testens nøjagtighed betydeligt. I dette kapitel beskrives de testværktøjer og automatiseringstilgange, der anvendes af testanalytikere. Det bør bemærkes, at testanalytikerne arbejder sammen med udviklerne, testautomatiseringsteknikerne og de tekniske testanalytikere om at skabe testautomatiseringsløsninger. Testanalytikere involveres i særdeleshed i nøgleordsdrevet automatisering og udnytter deres erfaringer med systemfunktionalitet og inden for branchen.

Yderligere oplysninger om testautomatiseringen og testautomatiseringsteknikerens rolle kan findes i ISTQB® Advanced Level Test Automation Engineer syllabus [ISTQB_ALTAE_SYL].

6.2 Nøgleordsdrevet Test

Nøgleordsdrevet test er en af de førende tilgange til testautomatisering, og den involverer testanalytikeren, der leverer de primære input: nøgleord og data.

Nøgleord (som også kaldes ”action words”) bliver primært, men ikke udelukkende anvendt til at repræsentere interaktioner med et system på højt niveau (f.eks. “annuller ordren”). Hvert nøgleord repræsenterer typisk et antal detaljerede interaktioner mellem en aktør og det system, der testes. Sekvenser af nøgleord (herunder relevante testdata) anvendes til at specificere testcases [Buwalda02].

I testautomatisering implementerer man et nøgleord som et eller flere eksekverbare testscripts. Værktøjerne læser testcases, der er skrevet i sekvenser af nøgleord, og som kalder de testscripts, der implementerer funktionaliteten. Scriptet implementeres modulært, så det er nemt at knytte det til specifikke nøgleord. Det er nødvendigt med færdigheder i programmering for at kunne implementere modulære scripts.

Primære fordele ved nøgleordsdrevet test:

- Nøgleord, der har med en særlig applikation eller branche at gøre, kan defineres af brancheeksperter. Det kan gøre opgaven med specifikation af testcasen mere effektiv
- En person, der primært har brancheekspertise, kan drage fordel af automatiseret afvikling af testcases (når nøgleordene er blevet implementeret som scripts) uden at skulle forstå den underliggende automatiseringskode
- Anvendelse af modulære skriveteknikker gør det nemt for testautomatiseringsteknikeren at vedligeholde testcases, når der skal ske ændringer af funktionaliteten og af softwarens brugergrænseflade [Bath14]
- Testcases kan specificeres uafhængigt af deres implementering

Generelt er det testanalytikernes opgave at oprette og vedligeholde data for nøgleord/action words. De skal forstå, at det stadig er nødvendigt at udvikle scripts for at kunne implementere nøgleord. Når de nøgleord og data, der skal anvendes, er blevet defineret, skal den testautomatiseringsansvarlige (f.eks. den tekniske testanalytiker eller testautomatiseringsteknikeren) oversætte forretningsprocessens nøgleord og lavniveaushandlinger til automatiserede testscripts.

Da nøgleordsdrevet test generelt afvikles under systemtesten, kan udvikling af koden påbegyndes så tidligt som i testdesignet. I et iterativt miljø, og især når der anvendes en integration/kontinuerlig implementering, vil testautomatiseringsudvikling være en kontinuerlig proces.

Når input-nøgleordene og data er oprettet, har testanalytikeren ansvaret for at afvikle de nøgleordsdrevne testcases og for at analysere enhver afvigelse, der kan opstå.

Når anomalier opdages, skal testanalytikeren assistere med at undersøge årsagen til afvigelsen for at afgøre, om defekten ligger i nøgleordene, inputdataene, selve testautomatiseringscriptet eller i den applikation, der testes. Generelt er første skridt af fejlfindingen at afvikle samme test med samme data manuelt for at se, om fejlen findes i selve applikationen. Hvis den ikke viser fejl, skal testanalytikeren gennemgå testsekvensen, der leder til fejlen for at afgøre, om problemet har fundet sted i et tidligere trin (måske ved at introducere

ukorrekte inputdata), men ikke vist sig før senere i databehandlingen. Hvis testanalytikeren ikke er i stand til at afgøre fejlens årsag, bliver fejlfindingsoplysningerne overleveret til den tekniske testanalytiker eller udvikleren for yderligere analyse.

6.3 Typer af testværktøjer

Meget af testanalytikerens arbejde kræver den mest effektive brug af værktøjer. Effektiviteten øges af de følgende:

- Forståelse af, hvilke værktøjer der skal anvendes
- Forståelse af, hvilke værktøjer der øger testindsatsens effektivitet (f.eks. ved at hjælpe med at give en bedre dækning inden for den tildelte tid)

6.3.1 Værktøjer til testdesign

Værktøjer til testdesign anvendes for at hjælpe med at oprette de testcases og testdata, der skal anvendes til test. Disse værktøjer kan fungere ud fra specifikke kravdokumentformater, modeller (f.eks. UML) eller de input, som testanalytikeren har leveret. Testdesignværktøjet er ofte designet og bygget til at fungere med særlige formater og værktøjer som specifikke kravstyringsværktøjer.

Testdesignværktøjer leverer oplysninger, som testanalytikeren kan anvende, når det skal afgøre, hvilke typer af tests, der er nødvendige for at indhente det specifikke målniveau for dækningen, tillid til systemet eller risikominimering af produktrisici. F.eks. genererer (og viser) værktøjer til klassifikationstræer de kombinationer, der er nødvendige for at opnå fuld dækning, baseret på det valgte dækningskriterium. Denne oplysning kan anvendes af testanalytikeren til at afgøre, hvilke testcases, der skal afvikles.

6.3.2 Værktøjer til forberedelse af testdata

Værktøjer til forberedelser af testdata kan give de følgende fordele:

- Analysere et dokument f.eks. kravdokumentet eller endda kildekoden for at afgøre de påkrævede data i løbet af testen for at opnå et særligt dækningsniveau
- Tag et datasæt fra et produktionssystem og ryd eller anonymiser for enhver form for persondata, mens dataenes interne integritet opretholdes. De ryddede data kan derefter anvendes til test uden risiko for en sikkerhedslækage eller misbrug af persondata. Det er særligt vigtigt, når der kræves store mængder af realistiske data, og der findes sikkerheds- og databeskyttelsesrisici
- Generér syntetiske testdata fra et givet sæt af inputparametre (f.eks. til anvendelse i tilfældige tests). Nogle værktøjer kan analysere databasens struktur for at afgøre, hvilke input der kræves af testanalytikeren

6.3.3 Værktøjer til automatiseret testafvikling

Testafviklingsværktøjer anvendes af testanalytikeren på alle testniveauer for at afvikle automatiserede tests og for at tjekke testenes faktiske resultat. Formålet med at anvende testafviklingsværktøjer kan typisk være:

- At reducere omkostninger (i forhold til indsats og/eller tid)
- At afvikle flere tests
- At afvikle samme test i mange forskellige miljøer
- At gøre testafviklingen mere gentagelig
- At afvikle tests, der ville have været umulige at afvikle manuelt (f.eks. store datavalideringstests)

Disse målsætninger har ofte overlappende hovedmål i forhold til at øge dækningen, mens omkostningen reduceres.

Investeringsafkastet for testafviklingsværktøjer er generelt højest, når man automatiserer regressionstests pga. det lave niveau af forventet vedligeholdelse og den gentagne afvikling af testene. Automatisering af smoketests kan også være en effektiv anvendelse af automatisering pga. den hyppige anvendelse af tests,

behovet for et hurtigt testresultat - og til trods for, at vedligeholdelsesomkostninger kan være højere - evnen til at have en automatiseret metode til at evaluere et nyt build i et kontinuerligt integrationsmiljø.

Testafviklingsværktøjer bliver typisk anvendt i løbet af system- og integrationstesten. Nogle værktøjer, særligt API-testværktøjer, kan også anvendes i komponenttests. Hvis man udnytter værktøjerne, hvor de er mest anvendelige, kan det hjælpe med at øge investeringsafkastet.

7. Henvisninger

Standarder

- [ISO25010] ISO/IEC 25010 (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models, kapitel 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 NotationTM, Version 1.3, December 2019; url: http://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, kapitel 1

ISTQB® dokumenter

- [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

Bøger og artikler

- [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, Dorothy Graham, “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] 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

Andre kilder (der ikke er direkte henvist til i denne syllabus)

De følgende henvisninger peger på oplysninger, der er tilgængelige på internettet og andetsteds. Selvom disse henvisninger blev gennemset, da denne Advanced Level syllabus blev frigivet, er ISTQB® ikke ansvarlig, hvis henvisningen ikke er tilgængelig længere.

- Kapitel 3
  - Czerwonka, Jacek: www.pairwise.org
  - God oversigt over forskellige defekt taksonomier: www.testingeducation.org/a/bugtax.pdf
  - Heuristic Risk-Based Testing af James Bach
  - Exploring Exploratory Testing, Cem Kaner og 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. Bilag A

Den følgende tabel er afledt af den komplette tabel, der står i ISO 25010. Den fokuserer kun på de kvalitetsegenskaber, der dækkes i testanalytikerens syllabus og sammenligner de termer, der anvendes i ISO 9126 (som anvendes i 2012-versionen af denne syllabus) med dem fra den nyere ISO 25010 (som anvendes i denne version).

ISO/IEC 25010	ISO/IEC 9126-1	Noter
Funktionel egnethed	**Funktionalitet**	
Funktionel fuldstændighed		
Funktionel korrekthed	Nøjagtighed	
Funktionel hensigtsmæssighed	Egnethed	
	Tværoperationalitet	Flyttet til kompatibilitet
Brugervenlighed		
Passende genkendelighed	Forståelighed	Nyt navn
Læringsegnethed	Læringsegnethed	
Operationalitet	Operationalitet	
Beskyttelse mod brugerfejl		Nye underegenskaber
Brugergrænsefladens æstetik	Attraktivitet	Nyt navn
Tilgængelighed		Nye underegenskaber
Kompatibilitet		Ny definition
Interoperabilitet		
Sameksistens		Dækket i Teknisk testanalytiker

9. Indeks