Certificeret Tester

Advanced Test Analyst Pensum

Dansk version 2012

International Software Test Qualifications Board

Advanced Niveau Test Analyst underarbejdsgruppe: Judy McKay (Chair), Mike Smith, Erik Van Veenendaal; 2010-2012.

Revisionshistorie

Version	Dato	Bemærkning
ISEB v1.1	04SEP01	ISEB Practitioner Syllabus
ISTQB 1.2E	SEP03	ISTQB Advanced Level Syllabus from EOQ-SG
V2007	12OCT07	Certified Tester Advanced Level syllabus version 2007
D100626	26JUN10	Incorporation of changes as accepted in 2009, separation of chapters for the separate modules
D101227	27DEC10	Acceptance of changes to format and corrections that have no impact on the meaning of the sentences.
D2011	23OCT11	Change to split syllabus, re-worked LOs and text changes to match LOs. Addition of BOs.
Alpha 2012	09MAR12	Incorporation of all comments from NBs received from October release.
Beta 2012	07APR12	Incorporation of all comments from NBs received from the Alpha release.
Beta 2012	07APR12	Beta Version submitted to GA
Beta 2012	08JUN12	Copy edited version released to NBs
Beta 2012	27JUN12	EWG and Glossary comments incorporated
RC 2012	15AUG12	Release candidate version - final NB edits included
GA 2012	19OCT12	Final edits and cleanup for GA release
DSTB v1.0	5. juni 2014	Dansk udgave frigivet

Indholdsfortegnelse

Bidragsydere

Dette dokument er skrevet af et hovedteam i International Software Testing Qualifidations Board Advanced Niveau underarbejdsgruppen Advanced Test Analyst: Judy McKay (Chair), Mike Smith, Erik van Veenendaal.

Hovedteamet takker reviewteamet og de nationale boards for deres forslag og input.

Da Advanced Niveau pensummet var skrevet færdig bestod Advanced Niveau arbejdsgruppen af følgende medlemmer (i alfabetisk rækkefølge):

Graham Bath, Rex Black, Maria Clara Choucair, Debra Friedenberg, Bernard Homès (Vice Chair), Paul Jorgensen, Judy McKay, Jamie Mitchell, Thomas Mueller, Klaus Olsen, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Jan Sabak, Hans Schaefer, Mike Smith (Chair), Geoff Thompson, Erik van Veenendaal, Tsuyoshi Yumoto.

Følgende personer deltog i review arbejdet og anden kommentering af samt afstemninger om indholdet:

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

Dokumentet blev formelt frigivet af ISTQB®'s generalforsamling den 19. oktober 2012.

Følgende personer deltog i bearbejdningen til dansk:

- Oversættelse: Anne Mette Jonassen Hass
- Review: Lisbeth Hylke Thomsen, Agnete K. Jensen m.fl.
- Redigering: Mette Bruhn-Pedersen, Ebbe Munk og Klaus Olsen

0 Introduktion til pensum

0.1 Formålet med dette dokument

Dette pensum er grundlaget for den internationale certificering i softwaretest for testanalytikere på Advanced niveau. ISTQB® stiller dette pensum til rådighed som følger:

1. Til nationale boards med henblik på oversættelse til deres lokale sprog og til at akkreditere kursusudbydere. Nationale boards kan tilpasse beskrivelsen af pensum til deres specifikke sprogbehov og ændre referencer for at tilpasse sig til deres lokale publikationer
2. Til eksamensboards, for at skrive eksamensspørgsmål på det lokale sprog i henhold til læringsmålene i pensummet
3. Til kursusudbydere for at producere kursusmateriale og finde passende undervisningsmetoder
4. Til certificeringskandidater som forberedelse til eksamen - som del af et kursus eller individuelt
5. Til det internationale software- og systemudviklingssamfund for at fremme software- og systemtestprofessionen, og som grundlag for bøger og artikler.

ISTQB® kan give andre tilladelse til at bruge beskrivelsen af pensum til andre formål, hvilket kræver en skriftlig tilladelse.

0.2 Overblik

Det videregående niveau er sammensat af tre pensa:

- Testmanager
- Testanalytiker
- Teknisk testanalytiker.

Det videregående niveaus overbliksdokument [ISTQB_AL_OVIEW] indeholder følgende information:

- Forretningsværdien af hvert pensum
- Sammendrag af hvert pensum
- Relationer mellem pensa
- Beskrivelse af kognitive niveauer (K-niveauer)
- Bilag.

0.3 Eksaminerbare læringsmål

Læringsmålene understøtter forretningsværdierne og bruges til at udforme eksamen til opnåelse af Advanced Test Analyst certificering. Generelt er alle dele af dette pensum eksaminerbart på K1 niveau. Det betyder, at kandidaten skal genkende og huske en term eller et koncept. Læringsmålene på K2, K3 og K4 niveau er vist i starten af det relevante kapitel.

0.4 Dansk oversættelse

Som nævnt ovenfor i afsnit 0.1 kan nationale boards tilpasse beskrivelsen af pensum til deres specifikke sprogbehov. Det har vi i DSTB valgt at gøre i nogle passager af dette pensum for at gøre teksten mere læsbar i den danske oversættelse.

1 Testprocessen - 300 min.

Nøgleord

Konkret testcase, slutkriterier, højniveau testcase, logisk testcase, lavniveau testcase, testkontrol, test design, testafvikling, testimplementering, testplanlægning

Læringsmål for testprocessen

1.2 Test i softwareudviklingens livscyklus

TA-1.2.1 (K2) Forklar hvordan og hvorfor tidspunkt og niveau for involvering af testanalytikeren afhænger af den valgte livscyklusmodel.

1.3 Testplanlægning, overvågning og kontrol

TA-1.3.1 (K2) Opsummér de aktiviteter som testanalytikeren udfører for at støtte planlægning og kontrol af testen.

1.4 Testanalyse

TA-1.4.1 (K4) Analyser et givet scenarie, herunder projektbeskrivelse og livscyklusmodel, for at finde passende opgaver for testanalytikeren i analyse- og designfaserne.

1.5 Testdesign

TA-1.5.1 (K2) Forklar hvorfor testbetingelserne bør være forstået af interessenterne.

TA-1.5.2 (K4) Analyser et projektscenarie for at fastlægge passende brug af lavniveau (konkrete) og højniveau (logiske) testcases.

1.6 Implementering af testen

TA-1.6.1 (K2) Beskriv de typiske slutkriterier for testanalyse og testdesign og forklar hvordan opfyldelsen af disse kriterier påvirker arbejdsindsatsen for testimplementeringen.

1.7 Afvikling af testen

TA-1.7.1 (K3) For et givet scenarie: Bestem fremgangsmåden og de hensyn, der bør tages ved testafviklingen.

1.8 Evaluering og rapport ved afslutning

TA-1.8.1 (K2) Forklar hvorfor det er vigtigt at sørge for præcis information om status for testcaseafviklingen.

1.9 Testlukning

TA-1.9.1 (K2) Giv eksempler på de arbejdsprodukter, som en testanalytiker bør levere i forbindelse med testlukningen.

1.1 Introduktion

I pensum for Foundation-niveauet omfatter den fundamentale testproces følgende aktiviteter:

- Planlægning, overvågning og kontrol
- Analyse og design
- Implementering og afvikling
- Evaluering af slutkriterier og rapportering
- Testlukningsaktiviteter.

På Advanced niveau er nogle af disse aktiviteter behandlet hver for sig for at give ekstra detaljering og optimering af processerne, bedre tilpasning til softwareudviklingslivscyklus, og for at understøtte mere effektiv testovervågning og kontrol. Aktiviteterne på dette niveau anses for at være som følger:

- Planlægning, overvågning og kontrol
- Analyse
- Design
- Implementering
- Afvikling
- Evaluering af slutkriterier og rapportering
- Testlukningsaktiviteter.

Disse aktiviteter kan implementeres sekventielt eller nogle kan implementeres i parallel, f.eks. kunne design foretages parallelt med implementering (f.eks. udforskende test). Testanalytikerens primære fokusområde er at planlægge de rigtige tests og testcases, designe dem og afvikle dem. Selv om det er vigtigt at forstå andre trin i testprocessen, udføres hovedparten af testanalytikerens arbejde normalt i testprojektets analyse-, design-, implementerings- og afviklingsaktiviteter.

Videregående testere står over for en række udfordringer, når de introducerer de forskellige testaspekter beskrevet i dette pensum i deres egne organisationer, teams og opgavekontekst. Det er vigtigt at tage hensyn til de forskellige softwareudviklingslivscykler så vel som til typen af det system, der testes, da disse faktorer kan påvirke tilgangen til test.

1.2 Test i softwareudviklingens livscyklus

Den langsigtede livscyklustilgang til test bør overvejes og defineres som en del af teststrategien. Testanalytikeren involveres på forskellige tidspunkter i forskellige livscykler, og mængden af involvering, den nødvendige tid, den tilgængelige information og forventningerne kan også være meget forskellige. Da testprocessen ikke foregår i isolation, skal testanalytikeren være opmærksom på de tidspunkter, hvor information kan leveres til andre relaterede organisatoriske områder som f.eks.:

- Kravudarbejdelse og kravstyring – kravreview
- Projektledelse – information om tidsplanlægning
- Konfigurationsstyring og ændringsstyring – verifikationstest af builds, versionskontrol
- Softwareudvikling – forudse, hvad der kommer, og hvornår
- Softwarevedligeholdelse – defektstyring, behandlingstid (dvs. tid fra defektopdagelse til defektløsning)
- Teknisk support – præcis dokumentation af workarounds
- Produktion af teknisk dokumentation (f. eks., databasedesignspecifikationer) – input til disse dokumenter så vel som teknisk review af dokumenterne.

Testaktiviteterne skal være afstemt med den valgte udviklingslivscyklusmodel, som kan være sekventiel, iterative eller inkrementel. For eksempel, kunne ISTQB®s fundamentale test proces anvendt på systemtestniveau i den sekventielle V-model, afstemmes som følger:

- Systemtestplanlægning sker samtidig med projektplanlægning, og testkontrol fortsætter indtil systemtestafviklingen og lukningen er gennemført
- Systemtestanalyse og design sker samtidig med kravspecifikation, system- og overordnet (højniveau) designspecifikation, og komponent- (lavniveau) designspecifikation
- Implementering af systemtestmiljø (f. eks. testbænk eller testrig) kan starte under systemdesign, selvom størstedelen af arbejdet typisk vil ske samtidig med kodning og komponenttest. Aktiviteter vedrørende systemtestimplementering strækker sig ofte til få dage før starten på systemtestafviklingen
- Systemtestafvikling begynder, når alle startkriterierne for systemtesten er opfyldt (eller frafaldet), hvilket typisk betyder, at som minimum komponenttest og ofte også komponentintegrationstest er gennemført. Systemtestafvikling fortsætter til systemtestens slutkriterier er opfyldt
- Evaluering af systemtestens slutkriterier og rapportering af resultaterne for systemtesten sker under hele systemtestafviklingen, generelt med større frekvens og betydning efterhånden som projektets deadlines nærmer sig
- Systemtestlukning sker efter at systemtestens slutkriterier er opfyldt og systemtestafvikling er erklæret gennemført, selv om det somme tider kan blive forsinket til efter at accepttest er ovre og alle projektaktiviteter er gennemført.

Iterative og inkrementelle modeller følger ofte ikke den samme rækkefølge af aktiviteter og kan udelukke nogle aktiviteter. For eksempel kan en iterativ model anvende et reduceret sæt af standard testprocesserne i hver iteration. Analyse og design, implementering og afvikling, samt evaluering og rapportering kan foretages for hver iteration, hvorimod planlægning ofte sker i begyndelsen af projektet og afslutningsrapportering sker til sidst. I et agilt projekt er det almindeligt at bruge en mindre formaliseret proces og langt tættere samarbejde, hvilket muliggør at ændringer lettere kan håndteres internt i projektet. Fordi agile er en “letvægts-” proces, er der ikke så omfattende testdokumentation, men i stedet en hurtigere kommunikationsmetode, som f.eks. “stand-up” møder (kaldt “stand up” fordi de er meget korte, typisk 10-15 minutter, så ingen behøver at sidde ned og alle kan forblive involveret).

Blandt de forskellige livscyklusmodeller kræver agile projekter den tidligste involvering af testanalytikeren. Testanalytikeren skal forvente at blive involveret fra projektets initiering og arbejde sammen med udviklerne, når de udfører deres allerførste arkitektur- og designarbejde. Statisk test er måske ikke formaliseret, men sker kontinuert efterhånden som software udvikler sig. Involvering forventes gennem hele projektet og testanalytikeren bør være til rådighed for teamet. På grund af denne dybe involvering er medlemmer af agile teams ofte dedikeret til enkelte projekter og er fuldt involveret i alle aspekter af projektet.

Iterative/inkrementelle modeller rækker fra den agile tilgang, hvor ændringer forventes efterhånden som softwaren udvikler sig, til iterative/inkrementelle udviklingsmodeller, der eksisterer inden for en V-model (nogle gang kaldet indlejret iterativ). Når der er tale om en indlejret iterativ model, bør testanalytikeren forvente at blive involveret i de almindelige planlægnings- og designaspekter, men vil derefter overgå til en mere interaktiv rolle efterhånden som softwaren udvikles, testes, ændres og sættes i drift.

Uanset hvilken softwareudviklingslivscyklus, der anvendes, er det vigtigt for testanalytikeren at forstå forventningerne mht. involvering så vel som tidspunktet for involveringen. Der er mange hybridmodeller i brug, som f.eks. den iterative inden for en V-model som beskrevet ovenfor. Testanalytikeren må ofte finde den mest effektive rolle og arbejde mod den, snarere end at hænge sig i definitionen af en anvendt model til at styre, hvad der er det rette tidspunkt for involvering.

1.3 Testplanlægning, overvågning og kontrol

Dette afsnit fokuserer på processerne for planlægning, overvågning og kontrol af test.

1.3.1 Testplanlægning

Testplanlægning sker for det meste ved starten af testarbejdet. Den omfatter identifikation og planlægning af alle de aktiviteter og ressourcer, der er nødvendige for at opfylde missionen og formålene identificeret i teststrategien. I løbet af testplanlægningen er det vigtigt for testanalytikeren at overveje og planlægge følgende i samarbejde med testmanageren:

- Vær sikker på, at testplanerne ikke er begrænset til funktionel test. Alle typer af test bør overvejes i testplanen og tidsplanlægges i henhold hertil. For eksempel kan testanalytikeren være ansvarlig for brugbarhedstest ud over funktionel test. Den type test skal også være dækket i et testplandokument
- Gennemgå testestimater med testmanageren og den øvrige ledelse og sørg for at der er afsat tilstrækkelig tid til anskaffelse og validering af testmiljøet
- Planlæg konfigurationstest. Hvis mange typer processorer, operativsystemer, virtuelle maskiner, browsere og forskellige ydre enheder kan kombineres til mange mulige konfigurationer, så planlæg at bruge testteknikker der vil give tilstrækkelig dækning af disse kombinationer.
- Planlæg test af dokumentationen. Brugere får både software og dokumentation. Dokumentationen skal være præcis for at være effektiv. Testanalytikeren skal afsætte tid til at verificere dokumentationen og kan have brug for at arbejde sammen med tekniske forfattere for at hjælpe med at forberede data, der skal bruges i screenshots og videoklip
- Planlæg at teste installationsprocedurerne. Installationsprocedurer, så vel som backup- og restoreprocedurer, skal testes tilstrækkeligt. Disse procedurer kan være mere kritiske end softwaren. Hvis softwaren ikke kan installeres, vil den ikke blive brugt overhovedet. Dette kan være svært at planlægge, da testanalytikeren ofte foretager den første test på et system, der er blevet præ-konfigureret uden at de endelige installationsprocedurer har været på plads
- Planlæg testen til at stemme overens med softwarelivscyklussen. Sekventiel afvikling af aktiviteter passer kun ind i de færreste tidsplaner. For mange aktiviteter er det ofte nødvendigt at udføre dem (i det mindste delvis) i parallel. Testanalytikeren skal være bevist om den valgte livscyklus og forventningerne til involvering under design, udvikling og implementering af softwaren. Dette omfatter også allokering af tid til gentest og regressionstest
- Afsæt tilstrækkelig tid til at finde og analysere risici sammen med det tværfaglige team. Selv om testanalytikeren som regel ikke er ansvarlig for at organisere risikostyringsworkshops, bør man forvente at blive aktivt involveret i aktiviteter.

Der kan være komplekse sammenhænge mellem testgrundlag, testbetingelser og testcases, sådan at der kan eksistere mange-til-mange relationer mellem disse arbejdsprodukter. Disse skal forstås for at testplanlægning og kontrol kan implementeres effektivt. Testanalytikeren er normalt den bedste til at bestemme disse relationer og til at arbejde for at skille afhængigheder så meget ad som muligt.

1.3.2 Testovervågning og kontrol

Skønt testovervågning og kontrol normalt er testmanagerens opgave, bidrager testanalytikeren med de måletal, der gør kontrollen mulig.

Forskellige kvantitative data bør indsamles gennem hele softwareudviklingslivscyklussen (f. eks. procentdel af planlægningsaktiviteter der er færdige, procentdel af dækning der er opnået, antal testcases der er bestået hhv. fejlet). I hvert tilfælde skal der defineres en baseline (dvs. standardreference) og herefter følges fremdrift i forhold til denne baseline. Mens testmanageren vil

være optaget af at strukturere og rapportere den indsamlede information, indsamler testanalytikeren informationen for hver metrik. Hver testcase der er færdig, hver fejlrapport der er skrevet, hver milepæl der er nået, vil slå igennem i de samlede projektmetrikker. Det er vigtigt, at den information, der opsamles i de forskellige opfølgningsværktøjer, er så præcise som muligt, så metrikkerne reflekterer virkeligheden.

Præcise metrikker giver ledelsen mulighed for at overvåge og kontrollere et projekt. Der er mange anvendelsesmuligheder for præcise data. Eksempler:

- Et højt antal defekter i et bestemt område af softwaren kan vise, at der er brug for ekstra testarbejde i det område
- Oplysninger om kravdækning og risikodækning kan bruges til at fordele udestående opgaver og ressourcer
- Oplysninger om underliggende årsager kan bruges til at finde områder, hvor processerne kan forbedres.

Fremtidige projekter kan planlægges mere effektivt, når der findes gode data fra tidligere projekter. Ledelsen er afhængig af præcise data for at kunne kontrollere projektet og for at kunne kommunikere status videre til projektets interessenter. Testanalytikeren må sørge for at data er præcise, rettidige og objektive.

1.4 Testanalyse

I testplanlægning defineres omfanget af testen. Testanalytikeren bruger denne omfangsbeskrivelse til at:

- Analysere testgrundlaget
- Identificere testbetingelserne.

For at testanalytikeren kan komme videre med sin testanalyse, skal følgende startkriterier være opfyldt:

- Der eksisterer et dokument, der beskriver testobjektet, og som kan bruges som testgrundlag
- Dette dokument er kommet igennem statisk test med fornuftige resultater, og er blevet opdateret efter behov efter den statiske test
- Der findes et fornuftigt budget og en fornuftig tidsplan for gennemførelse af det endnu ikke gennemførte testarbejde for testobjektet.

Testbetingelser findes typisk ved at analysere testgrundlaget og testens formål. Hvor dokumentationen er forældet eller helt mangler, kan testbetingelserne findes ved at tale med relevante interessenter (f. eks. i workshops eller under sprintplanlægning). Disse betingelser bruges derefter til at fastlægge, hvad der skal testes ved brug af testdesignteknikker fra teststrategi og/eller testplan.

Selv om testbetingelser normalt er specifikke i forhold til det, der testes, er der nogle bestemte hensyn som testanalytikeren må tage:

- Det er værd at definere testbetingelser på forskellige detaljeniveauer. Først finder man betingelserne på højt niveau for at definere generelle mål for testen, som f.eks. “funktionalitet af skærmbillede x”. Herefter identificeres mere detaljerede betingelser som grundlag for specifikke testcases, såsom “skærmbillede x afviser et kontonummer, der er et ciffer for kort i forhold til den korrekte længde”. Brug af denne type hierarkisk tilgang til at definere testbetingelser kan hjælpe med til at sikre, at dækningen er tilstrækkelig for højniveau objekter
- Hvis produktrisici er blevet defineret, skal de testbetingelser, der er nødvendige for at håndtere hver enkelt produktrisiko, identificeres og spores tilbage til risikoelementet.

Ved afslutning af testanalyseaktiviteterne bør testanalytikeren vide hvilke specifikke tests der skal designes for at dække behovet på området.

1.5 Testdesign

Testprocessen fortsætter ved at testanalytikeren designer de tests, der skal implementeres og afvikles. Der skal stadig arbejdes inden for den afgrænsning, der blev fastlagt i løbet af testplanlægningen. Testdesignprocessen omfatter:

- Bestem for hvilke testområder det er mest passende med testcases på lavt niveau (konkret) eller højt niveau (logisk)
- Bestem hvilke designteknikker for testcases, der giver den nødvendige testdækning
- Opret testcases der berører de fundne testbetingelser.

Man bør anvende de kriterier for prioritering, man fandt i løbet af risikoanalysen og testplanlægningen gennem hele processen: analyse, design, implementering og afvikling.

Et af startkriterierne for testdesign kan være om der er adgang til de værktøjer, der skal bruges under designarbejdet.

Når der designes tests, er det vigtigt at huske:

- Nogle testobjekter håndteres bedre ved kun at definere testbetingelserne, snarere end at gå dybere ind i ned at definere nedskrevne tests. I dette tilfælde bør testbetingelserne defineres, så de kan bruges som retningslinjer for den ubeskrevne test
- Bestående/ikke-bestående kriterier bør være klart defineret
- Tests bør designes, så de er forståelige for andre testere, ikke kun den, der skriver dem. Hvis den, der skriver testcases, ikke er den, der skal afvikle testen, vil det være nødvendigt at andre testere læser og forstår tidligere specificerede tests for at forstå testens formål og den relative vigtighed af testen
- Tests skal også være forståelige for andre interessenter som f.eks. udviklere, der skal gennemføre statisk test af testene, og auditøre, der måske skal godkende testene
- Tests bør designes til at omfatte alle interaktioner mellem softwaren og aktørerne (f. eks. slutbrugere og andre systemer), ikke kun de interaktioner, der sker gennem den synlige brugergrænseflade. Kommunikation mellem processer, batchafvikling og andre interrupts interagerer også med softwaren og kan indeholde defekter, så testanalytikeren skal designe tests for at nedsætte disse risici
- Tests bør designes til at teste grænseflader mellem de forskellige testobjekter.

1.5.1 Konkrete og logiske testcases

En af testanalytikerens opgaver er at bestemme, hvad der er de bedste typer af testcases i en givet situation. Konkrete testcases giver alle de specifikke informationer og procedurer, der er nødvendige for at testerne kan afvikle testcasene (inklusiv eventuelle datakrav) og verificere resultaterne. Konkrete testcases er anvendelige, når kravene er veldefinerede, når testmedarbejderne er mindre erfarne, og når ekstern verifikation af testene, som f.eks. audits, er krævet. Konkrete testcases giver fremragende gentagelighed (dvs. at en anden tester vil få det samme resultat), men kan også kræve en betydelig mængde vedligeholdelsesarbejde og har en tendens til at begrænse testerens kreativitet under afviklingen.

Logiske testcases giver retningslinjer for, hvad der skal testes, men tillader testanalytikeren at variere de faktiske data eller endda den procedure, der følges, når testen afvikles. Logiske testcases kan give bedre dækning end konkrete testcases, fordi de vil variere lidt hver gang de bliver afviklet. Dette leder også til tab i gentagelighed. Man må foretrække logiske testcases, hvis kravene ikke er veldefinerede,

hvis testanalytikeren har erfaring med både test og produktet, og hvis der ikke er krav om formel dokumentation af testen. Logiske testcases kan defineres tidligt i kravprocessen, også før kravene er veldefinerede. Logiske testcases kan bruges til at udvikle konkrete testcases, efterhånden som kravene bliver definerede og stabile. Så kan de logiske testcases erstattes af konkrete testcases.

1.5.2 Oprettelse af testcases

Testcases designes ved en trinvis bearbejdning af de fundne testbetingelser ved hjælp af de testdesign-teknikker, der er identificeret i teststrategien og/eller testplanen (se Kapitel 3). Testcases bør være gentagelige og verificerbare. De skal kunne spores tilbage til testgrundlaget som f. eks. krav - afhængigt af den valgte teststrategi.

Testcasedesign omfatter identifikation af:

- Formål
- Startbetingelser, som f.eks. enten projekt- eller mere specifikke testmiljøkrav og planer for, hvordan disse opfyldes, systemets tilstand osv.
- Testdatakrav (både inputdata for testcase, og data, der skal eksistere i systemet for at testcasene kan afvikles)
- Forventede resultater
- Slutbetingelser, som f.eks. påvirket data, systemets tilstand, udløsende faktorer for efterfølgende behandling osv..

Testcasenes detaljeringsniveau, som har betydning for både omkostningerne ved oprettelsen og gentagelighedsniveauet under afviklingen, bør defineres før testcasene oprettes. Færre detaljer i testcases giver testanalytikeren mere fleksibilitet når de afvikles og giver mulighed for at undersøge områder, der viser sig at være interessante. På den anden side giver færre detaljer som regel mindre gentagelighed.

Det er ofte en udfordring at bestemme det forventede resultat for en test. Det kan være besværligt at foretage en manuel beregning, og derfor kan det være bedre at finde eller skabe et automatisk orakel. Når en tester skal bestemme det forventede resultat, er han ikke kun optaget af det resultat, der er vist på skærmen, men også af slut-betingelser for data og miljø. Teoretisk set er det nemt at bestemme det korrekte resultat, hvis testgrundlaget er klart defineret. Imidlertid er testgrundlag ofte uklar, selvmodsigende, uden dækning af nøgleområder - eller mangler helt. Så må testanalytikeren selv skaffe sig viden om området. Komplekse interaktioner for komplekse stimuli og reaktioner kan desuden gøre det vanskeligt at bestemme, også hvor testgrundlaget er klart defineret. Derfor er det en afgørende fordel at have adgang til et testorakel. Det giver meget lidt værdi at afvikle testcases i situationer hvor man ikke kan bedømme om resultatet er korrekt. Det kan være grunden til både fejlagtige hændelsesrapporter og falsk tillid til systemet.

De ovenfor beskrevne aktiviteter kan udføres for alle testniveauer selv om testgrundlagene vil være forskellige. For eksempel kan brugeracceptttesten primært være baseret på kravspecifikationen, usecases og beskrevne forretningsprocesser, mens komponenttest primært kan være baseret på lavniveau designspecifikationer, user stories og selve koden. Det er vigtigt at huske på, at disse aktiviteter forekommer gennem alle testniveauerne selv om formålene med testene kan være forskellige. For eksempel er funktionel test på unitniveau designet for at en specifik komponent leverer den funktionalitet, der er specificeret i det detaljerede design for komponenten. Funktionel test på integrationsniveauet verificerer, at komponenterne interagerer med hinanden og leverer funktionalitet gennem deres interaktion. På systemniveauet bør end-to-end funktionalitet være formålet med testen. Når der analyseres og designes tests, er det vigtigt at huske testens målniveau så vel som formålet med testen. Dette hjælper med til at bestemme det nødvendige detaljeringsniveau så vel som hvilke værktøjer, der kan være behov for (f. eks., drivere og stubbe på komponenttestniveauet).

Under udviklingen af testbetingelser og testcases skaber man typisk noget dokumentation, dvs. et testarbejdsprodukt. Det varierer betydeligt, hvordan testarbejdsprodukter er dokumenteret. Det bliver påvirket af:

- Projektrisici (hvad skal/skal ikke dokumenteres)
- Den ekstra værdi, som dokumentationen giver til projektet
- Standarder, der skal følges og/eller bestemmelse, der skal opfyldes
- Den anvendte livscyklusmodel (f.eks. stiler en agil tilgang mod ”lige nok” dokumentation)
- Krav til sporing fra testgrundlaget til testanalyse og design.

Testanalyse og –design forholder sig til testobjektets/-objekternes kvalitetskarakteristikker afhængigt af testens omfang. ISO 25000 standarden [ISO25000] (som afløser ISO 9126) er en anvendelig reference. Når der testes hardware/software systemer, kan flere karakteristikker være gældende.

Testanalyse- og testdesignprocesserne kan forbedres ved at sammenflette dem med statisk test og statisk analyse. Faktisk fungerer gennemførslen af testanalysen og testdesignet ofte som en form for statisk test, fordi der kan findes problemer i basisdokumenterne under denne proces. Testanalyse og testdesign baseret på kravspecifikationen er en glimrende måde at forberede et kravreviewmøde på. Det kræver forståelse af kravene og evne til at vurdere opfyldelse af kravet hvis testanalytikeren læser krav for at bruge dem til at oprette tests. Denne aktivitet afdækker ofte krav, der er uklare, er utestbare eller ikke har definerede acceptkriterier. Tilsvarende bør testarbejdsprodukter som f.eks. testcases, risikoanalyser og testplaner underkastes statisk test.

Blandt andet projekter der følger en agil livscyklus, kan have minimalt dokumenterede krav. Kravene kan have form af user stories, som beskriver små, men demonstrerbare bidder af funktionalitet. En user story bør indeholde en definition af acceptkriterierne. Hvis softwaren er i stand til at demonstrere, at den har opfyldt acceptkriterierne, anses den normalt for at være klar til integration med anden færdig funktionalitet, eller kan allerede være blevet integreret for at demonstrere dens funktionalitet.

Under testdesign kan de nødvendige detaljerede test-infrastrukturkrav blive defineret, selvom disse i praksis måske ikke kan færdiggøres før under testimplementeringen. Det må huskes, at test-infrastruktur omfatter mere end testobjekter og testdelprodukter. For eksempel kan infrastrukturkravene omfatte lokaler, udstyr, personer, software, værktøjer, ydre enheder, kommunikationsudstyr, brugerrettigheder og alle andre ting, der er nødvendige for at afvikle testene.

Slutkriterierne for testanalyse og testdesign vil variere afhængig af projektparametrene, men alle emner omtalt i disse to afsnit bør overvejes i forbindelse med definition af slutkriterier. Det er vigtigt, at kriterierne er målbare, og at den information og forberedelse, der er nødvendig for de efterfølgende trin, er leveret.

1.6 Testimplementering

Testimplementering er realisering af testdesignet. Dette omfatter oprettelse af automatiserede tests, organisering af tests (både manuelle og automatiserede) i afviklingsrækkefølge, færdiggørelse af testdata og testmiljøer, og oprettelse af en tidsplan for testafvikling inklusiv resurseallokering, for at gøre det muligt at begynde at afvikle testcasene. Det omfatter også kontrol af eksplicitte og implicitte startkriterier for testniveauet og sikring af at slutkriterierne for det foregående trin i processen er opfyldt. Hvis slutkriterierne er blevet sprunget over for et testniveau eller for et trin i testprocessen, bliver implementeringen sandsynligvis forsinket med overarbejde og et dårligt resultat. Derfor er det vigtigt at alle slutkriterier er opfyldt før testimplementeringen begynder.

Der kan være mange overvejelser, når man skal fastlægge rækkefølgen for afviklingen. I nogle tilfælde kan det give mening at organisere testcasene i testsuiter, dvs. grupper af testcases. Dette kan hjælpe organiseringen af testen, sådan at relaterede testcases afvikles sammen. Hvis der anvendes en risikobaseret teststrategi, kan risikoprioriteringsordenen diktere afviklingsrækkefølgen af testcasene. Der kan være andre faktorer, der bestemmer rækkefølgen, som f.eks. adgang til personer, udstyr, data og den funktionalitet, der skal testes. Det er ikke usædvanligt for kode at blive frigivet i sektioner, og testarbejdet skal koordineres med den rækkefølge i hvilken softwaren bliver tilgængelig for test. Specielt i inkrementelle livscyklusmodeller er det vigtigt for testanalytikeren at koordinere med udviklingsteamet for at softwaren bliver frigivet til test i en testbar rækkefølge. Under testimplementeringen bør testanalytikere færdiggøre og bekræfte den rækkefølge, som manuelle og automatiserede tests skal afvikles i, og omhyggeligt kontrollere for begrænsninger, der kan kræve, at tests afvikles i en særlig rækkefølge. Afhængigheder skal dokumenteres og checkes.

Detaljeniveauet og den tilhørende kompleksitet for arbejde udført under test implementering kan blive påvirket af detaljegraden af testcasene og testbetingelserne. I nogle tilfælde gælder regulerende regler, og tests bør kunne bevise overholdelse af relevante standarder som f.eks. the United States Federal Aviation Administration's DO-178B/ED 12B [RTCA DO-178B/ED-12B].

Som specificeret oven for er testdata nødvendige for test og i nogle tilfælde kan disse datasæt blive ganske store. Under implementering opretter testanalytikere input data og miljødata som skal læses ind i databaser og andre lignende opbevaringssteder. Testanalytikere opretter også data, som skal bruges i data-dreven automatiserede tests så vel som i manuel test.

Testimplementering drejer sig også om testmiljøet/miljøerne. På dette stadie bør miljøet/miljøerne være fuldt sat op og verificeret forud for testafvikling. Et "formålstjenligt" testmiljø er altafgørende, dvs. testmiljøet bør kunne give mulighed for afsløring af de defekter, der findes under den kontrollerede test, fungere normalt når der ikke opstår afvigelser og på passende vis afspejle produktions- eller slutbrugermiljøet for højere niveauer af test, hvis det er nødvendigt. Det kan være nødvendigt at foretage ændringer i testmiljøet under testafvikling afhængig af uventede ændringer, testresultater eller andre hensyn. Hvis der sker ændringer af miljøet under afviklingen, må man vurdere effekten af ændringerne på de tests, der allerede er blevet afviklet.

Under testimplementering skal testere sikre, at de, der er ansvarlige for oprettelse og vedligeholdelse af testmiljøet, er kendte og til rådighed, og at al testmaterialet og værktøjer til understøttelse af test og tilhørende processer er klar til brug. Dette omfatter konfigurationsstyring, fejlhåndtering og testlogning og -ledelse. Desuden skal testanalytikere verificere de procedurer, der opsamler data for vurdering af slutkriterier og rapportering af testresultater.

Det er klogt at anvende en velafbalanceret tilgang til testimplementering som fastlagt under testplanlægning. For eksempel blandes risikobaserede analytiske teststrategier ofte med dynamiske teststrategier. I dette tilfælde afsættes en procentdel af testimplementeringsarbejdet til test, som ikke følger foruddefinerede scripts (uspecificeret).

Uspecificeret test bør ikke være ad hoc eller uden mål, da det kan være uforudsigeligt i forhold til varighed og dækning, med mindre det er time-boxet og udstukket. Gennem årene har testere udviklet en række forskellige erfaringsbasered teknikker som f.eks. angreb, fejlgætning [Myers79], og udforskende test. Testanalyse, testdesign, og testimplementering forekommer stadig, men fortrinsvis under testafvikling.

Når sådanne dynamiske teststrategier følges, påvirker resultat af hver enkelt test analysen, designet, og implementeringen af de efterfølgende tests. Skønt disse strategier er letvægts og ofte effektive til at finde defekter, er der nogle ulemper. Disse teknikker kræver ekspertise hos testanalytikeren,

varigheden kan være svær at forudse, dækningen kan være svær at følge og gentagelighed kan mistes uden god dokumentation eller værktøjsunderstøttelse.

1.7 Afvikling af testen

Testafvikling begynder, når testobjektet er leveret og startkriterierne for testafvikling er tilfredsstillet (eller frafaldet). Tests bør afvikles ifølge den plan, der er defineret under testimplementeringen, men testanalytikeren bør have tilstrækkelig tid til at sikre dækning med yderligere, interessante testscenarier og virkemåder, der er observeret under test. Man bør beskrive enhver afvigelse, man finder under testen, også nødvendige varianter af testcases, der er nødvendige for at genskabe afvigelsen. Denne blanding af specificerede og uspecificerede (f.eks. udforskende) testteknikker hjælper med til at sikre mod manglende test på grund af mangler i den specificerede dækning. Det kan også være et middel mod pesticid-paradokset.

I centrum af testafviklingsaktiviteten står sammenligning af faktiske resultater med forventede resultater. Testanalytikere skal være opmærksomme på og fokusere på disse opgaver, ellers kan alt arbejdet med designe og implementere testen være spildt, når afvigelser ikke opdages (falsk-negativt resultat) eller korrekt virkemåde misklassificeres som ukorrekt (falsk-positiv resultat). Hvis det forventede og det faktiske resultat ikke er identisk, er der forekommet en hændelse. Hændelser skal undersøges nøje for at bestemme årsagen (hvilket kan eller ikke kan være en defekt i testobjektet), og for at samle data, der kan hjælpe med løsningen af hændelsen (se Kapitel 6 for flere detaljer om defektstyring).

Når man finder en afvigelse skal man vurdere testdokumentationen nøje for at sikre, at den er korrekt. Et testdokument kan være ukorrekt af forskellige grunde. Hvis det er ukorrekt, bør det rettes, hvorefter man gentager testen. Da ændringer i enten testgrundlag eller testobjekt kan gøre en testcase ukorrekt selv efter mange ganges vellykket afvikling, bør testere være opmærksomme på at de observerede resultater kan skyldes at testen er ukorrekt.

Man skal logge testresultaterne på passende vis under testafviklingen. Afviklede test, hvor resultaterne ikke er logget, kan man blive nødt til at gentage for at finde det korrekte resultat, dvs. ineffektivitet og forsinkelser. Bemærk, at tilstrækkelig logning kan være en løsning på problemer med dækning og gentagelighed ved testteknikker som f.eks. udforskende test. Eftersom testobjektet, testmaterialet og testmiljøer kan udvikle sig bør logningen identificere specifikke testede versioner så vel som specifikke miljøkonfigurationer. Testlogning giver en kronologisk fortegnelse over relevante detaljer om afviklingen af tests.

Logning af resultatet gælder både for de enkelte tests, for aktiviteter og for hændelser. Hver test skal identificeres entydigt, og dens status skal logges, efterhånden som testafviklingen skrider frem. Enhver hændelse, der påvirker testafviklingen, bør logges. Man bør også logge tilstrækkelig information til at kunne måle testdækningen og information så man kan dokumentere årsagen til eventuelle forsinkelser og afbrydelser i testen. Desuden bør man logge information til testkontrol, rapportering af testfremdriften, måling af slutkriterier og forbedring af testprocessen.

Logning varierer afhængigt af testniveauet og strategien. For eksempel hvis der forekommer automatiseret komponenttest, bør de automatiserede tests fremstille det meste af loginformationen. Hvis der forekommer manuel test vil testanalytikeren logge information om testafviklingen, ofte i et teststyringsværktøj, som vil følge testafviklingsinformationen. I nogle tilfælde, som det gjaldt for testimplementering, er mængden af testafviklingsinformation, der logges, påvirket af regulerende krav eller auditkrav.

I nogle tilfælde kan brugere eller kunder deltage i testafvikling. Dette kan være med til at give kunderne tillid til systemet - det forudsætter dog at brugertesten kun afslører få defekter. En sådan forudsætning er ikke holdbar på de tidlige testniveauer, men kan være det under accepttest.

Følgende er nogle særlige områder, der skal overvejes under testafvikling:

- Noter og udforsk "irrelevante" mærkværdigheder. Observationer eller resultater, der kan forekomme irrelevante, er ofte indikatorer på defekter, der som isbjerge lurer under overfladen
- Kontroller at produktet ikke gør noget, som det ikke bør gøre. Test er normalt en kontrol af at produktet gør hvad det forventes at gøre, men testanalytikeren skal også være sikker på, at produktet ikke fungerer forkert ved at gøre noget, som det ikke skulle gøre - for eksempel ekstra, uønskede funktioner
- Byg test-suiter og forvent, at de vil vokse og ændre sig over tid. Koden vil udvikle sig, og det vil være nødvendigt at implementere ekstra tests for at dække disse nye funktionaliteter, såvel som at regressionsteste andre områder af softwaren. Huller i test opdages ofte under afvikling. Opbygning af test-suiter er en kontinuert proces
- Tag noter med henblik på det næste testarbejde. Testopgaverne slutter ikke, når softwaren er leveret til brugere eller distribueret til markedet. En ny version eller release af softwaren vil højst sandsynligt blive fremstillet, så viden skal opbevares og overføres til den ansvarlige for det næste testarbejde
- Forvent ikke at genafvikle alle manuelle tests. Det er urealistisk at forvente at alle tests vil blive genafviklet. Hvis der er mistanke om et problem, bør testanalytikeren undersøge det, og notere det, hellere end at antage, at det vil blive fanget i en efterfølgende afvikling af testcasene
- Brug data i fejlopfølgningsværktøjet til at få inspiration til yderligere testcases. Overvej at oprette testcases for defekter, der blev opdaget under uspecificeret eller udforskende test og tilføj dem til regressions-testsuiten
- Find defekter før regressionstesten. Tiden er ofte begrænset for regressionstest og identifikation af defekter under regressionstest kan resultere i forsinkelser. Regressionstests finder generelt ikke en stor del af defekterne, mest fordi det er tests, der allerede er blevet afviklet (f.eks. for en tidligere version af den samme software), og defekter bør være fundet i disse tidligere afviklinger. Dette betyder ikke, at regressionstests bør fjernes fuldstændigt, kun at effektiviteten af regressionstests, i form af muligheden for af finde nye defekter, er lavere end i andre tests.

1.8 Evaluering af slutkriterier og rapportering

I forhold til testprocessen, betyder fremdriftsovervågning sikring af, at der bliver indsamlet den rette information til at opfylde rapporteringskravene. Dette omfatter måling af fremdrift mod færdiggørelse. Når slutkriterierne er defineret i planlægningsfasen, kan der være en nedbrydning i “skal” og “bør” kriterier. For eksempel kan kriterierne lyde, at der “ikke må være åbne Prioritet 1 eller Prioritet 2 defekter” og der “bør være 95 % bestået i gennemsnit over alle testcases”. I dette tilfælde skulle manglen på opfyldelse af et “skal” (”må ikke”) kriterium betyde, at slutkriterierne ikke er opfyldt, men en bestået rate på 93 % stadig kunne tillade projektet at fortsætte til næste niveau. Slutkriterier skal være klart definerede, så de kan vurderes objektivt.

Testanalytikeren er ansvarlig for at levere den information, som bruges af testmanageren til at vurdere fremdrift mod opfyldelse af slutkriterierne og for at sikre at data er præcise. Hvis, for eksempel, teststyringssystem leverer følgende statuskoder for gennemførte testcases:

- Bestået
- Ikke bestået
- Bestået, som undtagelse.

så må testanalytikeren være meget klar på hvad hver af disse betyder, og må bruge den givne status konsekvent. Betyder "bestået, som undtagelse" at der blev fundet en defekt, men at den ikke påvirker systemets funktionalitet? Hvad med en brugbarhedsdefekt, der betyder, at brugeren bliver forvirret? Hvis beståelsesraten er et "skal" afslutningskriterium, bliver medregning af en testcase som "ikke bestået" snarere end "bestået, som undtagelse" en kritisk faktor. Der må også tages hensyn til testcases, der er markeret som "ikke bestået", men hvor årsagen til afvigelsen ikke er en defekt (f.eks. hvis testmiljøet var konfigureret forkert). Hvis der er uklarhed om, hvordan metrikker bliver fulgt, eller om brugen af statusværdierne, skal testanalytikeren afklare dette med testmanageren, så informationen kan blive fulgt præcist og konsistent under hele projektet.

Testanalytikeren kan blive bedt om at skrive en statusrapport under testcyklerne så vel som at bidrage til den endelige rapport ved slutningen af testen. Dette kan både kræve indsamling af metrikker fra fejlhåndterings- og teststyringssystemer og en vurdering af testens overordnede dækning og fremdrift. Testanalytikeren må kunne bruge rapporteringsværktøjer og kunne give den efterspurgte information til testmanageren.

1.9 Testlukningsaktiviteter

Når det er besluttet, at testafviklingen er færdig, bør vigtige output fra testarbejdet indsamles og enten overdrages til den relevante person eller arkiveret. Tilsammen udgør dette testlukningsaktiviteterne. Testanalytikeren bør forvente at blive involveret i aflevering af arbejdsprodukter til dem, der har brug for arbejdsprodukterne. For eksempel bør kendte defekter, der er udskudt eller accepterede kommunikeres til dem, der skal bruge og supportere brugen af systemet. Tests og testmiljøer bør gives til dem der er ansvarlige for vedligeholdelsestest. Andre arbejdsprodukter kan være et regressionstestsæt (enten automatiseret eller manuelt). Information om testarbejdsprodukter skal være klart dokumenteret, inkl. passende links, og passende adgangsrettigheder må være givet.

Testanalytikeren bør også forvente at skulle deltage i retrospektive møder ("lessons learned"), hvor vigtige erfaringer (både fra internt i testprojektet og på tværs af hele softwareudviklingslivscyklussen) kan blive dokumenteret og planer udarbejdet til styrkelse af det "gode" og fjernelse eller i det mindste kontrol med det "dårlige". Testanalytikeren er en kyndig kilde til information på disse møder og skal deltage, hvis troværdig procesforbedringsinformation skal indsamles. Hvis det kun er testmanageren, der deltager i mødet, skal testanalytikeren overdrage den relevante information til testmanageren, så der kan præsenteres et præcist billede af projektet.

Arkivering af resultater, logs, rapporter og andre dokumenter og arbejdsprodukter i konfigurationsstyringssystemet skal også foretages. Denne opgave tilfalder ofte testanalytikeren, og det er en vigtig lukningsaktivitet, specielt hvis et fremtidigt projekt skal bruge denne information.

Mens det normalt er testmanageren, der bestemmer, hvilken information, der skal arkiveres, bør testanalytikeren også tænke over, hvilken information der kunne blive brug for, hvis projektet skulle blive startet igen en gang i fremtiden. Overvejelser om denne information ved projektets afslutning kan spare måneders arbejde, når projektet startes igen på et senere tidspunkt eller med et andet team.

2 Testanalytikerens bidrag til teststyring - 90 min

Nøgleord
Produktrisiko, risikoanalyse, risikoidentifikation, risikoniveau, risikostyring, risikoreduktion, risikobaseret test, testovervågning, teststrategi

Læringsmål for testanalytikerens bidrag til teststyring

2.2 Testfremdriftsovervågning og kontrol

TA-2.2.1 (K2) Forklar hvilke typer af information, der skal følges under testen af hensyn til den nødvendige overvågning og kontrol af projektet.

2.3 Distribueret, outsourcet og insourcet test

TA-2.3.1 (K2) Giv eksempler på gode kommunikationsmåder, når der arbejdes i et 24-timers testmiljø.

2.4 Testanalytikerens opgaver i risiko-baseret test

TA-2.4.1 (K3) I et givet projekt: Deltag i risikoidentifikation, gennemfør en risikovurdering og foreslå passende forholdsregler for risici.

2.1 Introduktion

Skønt der er mange områder, hvor testanalytikeren interagerer med og leverer data til testmanageren, koncentrerer dette afsnit sig om de specifikke områder af testprocessen, hvor testanalytikeren er en væsentlig bidragyder. Det forventes, at testmanageren vil efterspørge den ønskede information hos testanalytikeren.

2.2 Testfremdriftsovervågning og kontrol

Der er fem primære dimensioner, hvor testfremdrift overvåges:

- Produkt- (kvalitet) risici
- Defekter
- Test
- Dækning
- Tillid.

Produktrisici, defekter, test, og dækning kan og bliver ofte målt og rapporteret på særlige måder af testanalytikeren i projektet eller under drift. Tillid bliver som regel rapporteret subjektivt, selv om det kan måles vha. spørgeundersøgelser. Indsamling af den information, der er nødvendig for at støtte disse metrikker, er en del af testanalytikerens daglige arbejde. Det er vigtigt at huske, at præcisionen af disse er kritisk, da upræcise data vil give upræcise informationer om tendenser og kan lede til upræcise konklusioner. I værste tilfælde vil upræcise data resultere i forkerte ledelsesbeslutninger og skade testteamets troværdighed.

Når der anvendes en risikobaseret testtilgang, bør testanalytikeren følge:

- Hvilke risici, der er blevet reduceret ved test
- Hvilke risici, der anses for ikke at være reduceret.

Opfølgning på risikoreduktion foretages ofte med et værktøj, der også følger testafslutning (f. eks. teststyringsværktøjer). Dette kræver, at de fundne risici kan spores til testbetingelserne, som derefter kan spores til de testcases, der vil reducere risici - hvis testcasene afvikles med succes. På denne måde opdateres informationen vedr. risikoreduktionen automatisk efterhånden som testcasene opdateres. Dette kan gøres for både manuelle og automatiserede tests.

Opfølgning på defekter sker normalt vha. et defektværktøj. Når man registrerer en defekt, klassificerer man den også. Denne information kan bruges til at fremstille tendenser og grafer, der viser testens fremdrift og softwarens kvalitet. Der er mere om klassifikationen i kapitlet om defektstyring. Livscyklussen kan påvirke mængden af den defektdokumentation, der registreres, og den metode, der bruges til at registrere informationen.

Efterhånden som testen udføres, skal statusinformation om testcases registreres. Dette foretages som regel vha. et teststyringsværktøj, men kan også foretages manuelt, hvis det er nødvendigt. Information om testcase kan omfatte:

- Oprettelsesstatus (f. eks., designet, reviewet)
- Afviklingsstatus (f. eks. bestået, ikke bestået, blokeret, sprunget over)
- Afviklingsinformation (f. eks. dato og tid, testernavn, anvendte data)
- Afviklingsartefakter (f. eks., screen-shots, tilhørende logs).

Ligesom for de fundne risici, bør også testcases spores til de krav, de tester. Det er vigtigt for testanalytikeren at huske, at hvis testcase A er sporet til krav A, og det er den eneste testcase, der er sporet til det krav, vil krav A, når testcase A er afviklet og bestået, anses for at være opfyldt. Dette er

måske, måske ikke korrekt. I mange tilfælde er det nødvendigt med flere testcases for at test et krav helt igennem, men pga. begrænset tid, er kun en delmængde af disse tests faktisk oprettet. For eksempel, hvis det var nødvendigt med 20 testcases for en helt dækkende test af implementeringen af et krav, men kun 10 var oprettet og afviklet, så ville kravdækningsinformationen vise 100 % dækning, mens der i virkeligheden kun var opnået 50 % dækning. Præcis opfølgning på dækningen så vel som opfølgning på review-status af selve kravene kan bruges som en tillidsmåling.

Mængden (og detaljeniveauet) af information, der skal registreres afhænger af adskillige faktorer, heriblandt softwareudviklings-livscyklusmodellen. For eksempel vil der i agile projekter typisk blive registreret mindre statusinformation pga. den tætte interaktion mellem teamet og mere face-to-face kommunikation.

2.3 Distribueret, outsourcet og insourcet test

I mange tilfælde udføres alt testarbejdet ikke af et enkelt testteam sammensat af kollegaer til resten af projektteamet på et og samme sted som resten af projektteamet. Hvis testarbejdet foretages flere steder, kan dette testarbejde kaldes distribueret. Hvis der foretages et enkelt sted, kaldes det centraliseret. Hvis testarbejdet foretages på et eller flere steder af personer, der ikke er kollegaer med resten af projektteamet, og som ikke sidder sammen med projektteamet, kan det testarbejde kaldes outsourcet. Hvis testarbejdet fortages af personer, der sidder sammen med projektteamet, men ikke er kollegaer, kan det testarbejde kaldes insourcet.

Når der arbejdes i et projekt, hvor noget af testteamet er spredt over flere steder og/eller over flere firmaer, skal testanalytikeren være ekstra opmærksom på effektiv kommunikation og informationsoverførsel. Nogle organisationer arbejder på en ”24 timers test” model, hvor teamet i en tidszone forventes at overlade arbejdet til teamet i en anden tidszone, således at testen kan fortsætte døgnet rundt. Dette kræver speciel planlægning for de testanalytikere, der skal overdrage og modtage arbejdet. God planlægning er vigtig for a forstå ansvarsområder og er vital for at de rette informationer er tilgængelige.

Når mundtlig kommunikation ikke er muligt, må skriftlig kommunikation være tilstrækkelig. Det betyder, at e-mail, statusrapporter og effektiv brug af teststyrings- og defekthåndteringsværktøjer skal anvendes. Hvis teststyringsværktøjet tillader at tests knyttes til enkeltpersoner, kan det også fungere som tidsplanlægningsværktøj og en nem måde at overføre arbejde mellem personer på. Defekter, der er præcist registreret, kan overdrages til medarbejdere for den nødvendige opfølgning. Effektiv brug af disse kommunikationssystemer er vital for en organisation, der ikke kan forlade sig på daglig personlig interaktion.

2.4 Testanalytikerens opgaver i risikobaseret test

2.4.1 Overblik

Testmanageren har ofte det overordnede ansvar for etablering og vedligeholdelse af en risikobaseret teststrategi. Testmanageren vil normalt inddrage en testanalytiker for at få den risiko-baserede tilgang korrekt implementeret.

Testanalytikeren bør være aktivt involveret i de følgende risiko-baserede testaktiviteter:

- Risikoidentifikation
- Risikovurdering
- Risikoreduktion.

Disse aktiviteter gennemføres iterativt gennem hele projektets livscyklus for at håndtere opstående risici og ændrede prioriteter og for regelmæssigt at evaluere om kommunikere risikostatus.

Testanalytikere bør arbejde inden for det risiko-baserede testrammeværk, som testmanageren har etableret for projektet. De bør bidrage med deres viden om risici i forretningsdomænet, der er nedarvet i projektet, som f.eks. risici relateret til sikkerheds-, forretnings- og økonomiske overvejelser og politiske faktorer.

2.4.2 Risikoidentifikation

Risikoidentifikationsprocessen har størst sandsynlig for at finde det størst mulige antal væsentlige risici ved at få det bredest mulige udvalg af interessenter til at deltage. Fordi testanalytikere ofte har unik viden vedr. det specifikke forretningsområde for systemet under test, er de specielt velegnede til at gennemføre ekspertinterviews med domæneeksperterne og brugerne, gennemføre uafhængige vurderinger og fremme brugen af risikoskabeloner, gennemføre risiko-workshops, gennemføre brainstorm-møder med potentielle og nuværende brugere, definere test-checklister og bruge tidligere erfaringer med tilsvarende systemer eller projekter. Testanalytikeren bør arbejde tæt sammen med brugere og andre domæneeksperter for finde de forretningsrisici, der skal testes. Testanalytikeren kan også være til særlig hjælp ved identifikation af risicis potentielle effekter på brugerne og interessenterne.

Eksempler på risici der kan findes i et projekt:

- Problemer med præcision i beregninger
- Manglende brugbarhed, f.eks. manglende tastaturgenveje
- Læringsproblemer, f.eks. mangel på instruktioner til brugeren på steder i grænseflader, hvor der skal tages vigtige beslutninger.

Overvejelser om test af de specifikke kvalitetskarakteristikker er dækket i Kapitel 4 i denne pensumbeskrivelse.

2.4.3 Risikovurdering

Mens risikoidentifikation handler om at finde så mange relevante risici som muligt, er risikovurdering studiet af de fundne risici. Specielt kategorisering af hver enkelt risiko og bestemmelse af sandsynlighed og effekt ved hver risiko.

Bestemmelse af risikoniveauet omfatter vurdering af sandsynligheden for at noget sker og virkningen ved at det sker. Sandsynlighed for at det sker bliver almindeligvis fortolket som sandsynligheden for at et potentielt problem kan eksistere i systemet under test og vil blive observeret, når systemet er i produktion. Med andre ord opstår det potentielle problem fra en teknisk risiko. Den tekniske testanalytiker bør bidrage til at finde og forstå det potentielle tekniske risikoniveau for hver risiko, mens testanalytikeren bidrager til forståelse af den potentielle indvirkning på forretningen fra problemet, hvis det skulle indtræffe.

Virkningen ved forekomst er ofte fortolket som alvorsgraden af effekten på brugerne, kunderne eller andre interessenter. Med andre ord, det opstår fra en forretningsrisiko. Testanalytikeren bør bidrage til identifikation og vurdering af den potentielle forretningsdomæne- eller brugervirkning for hver risiko. Faktorer, der influerer på forretningsrisiko:

- Brugsfrekvens for den ramte feature
- Forretningstab
- Potential finansielle, økologiske eller sociale tab eller ansvar
- Civile eller kriminelle lovmæssige sanktioner

- Sikkerhedsovervejelser
- Bøder, tab af licens
- Mangel på fornuftige work-arounds
- Featurens synlighed
- Fejlens synlighed, der kan lede til negativ omtale og skade på image
- Tab af kunder.

Testanalytikeren skal bestemme niveauerne af forretningsrisici ud fra den tilgængelige risikoinformation ifølge retningslinjer fra testmanageren. Niveauerne kan klassificeres med ord (f. eks. lav, medium, høj) eller tal. Hvis der findes en objektiv metode til at måle risikoen på på en defineret skala, kan det være en ægte kvantitativ måling, men det er der som regel ikke tale om. Normalt er det vanskeligt at måle sandsynlighed og konsekvenser. Derfor bestemmer man normalt risikoniveauet som en kvalitativ bedømmelse.

Der kan knyttes tal til den kvalitative værdi, men det gør den ikke til en ægte kvantitativ måling. For eksempel kan testmanageren bestemme, at forretningsrisici skal kategoriseres med en værdi fra 1 til 10, hvor 1 er den højeste, og dermed mest risikofyldte, virkning på forretningen. Når sandsynligheden (vurderingen af den tekniske risiko) og virkningen (vurderingen af forretningsrisikoen) er blevet angivet, kan disse værdier ganges sammen for at bestemme det overordnede risikoniveau for hver risiko. Det overordnede niveau bruges derefter til at prioritere de risiko-reducerende aktiviteter. Nogle risikobaserede testmodeller, som f.eks. PRISMA® [vanVeenendaal12], kombinerer ikke risikoværdierne, således at testtilgangen kan forholde sig til de tekniske og forretningsmæssige risici hver for sig.

2.4.4 Risikoreduktion

I løbet af projektet bør testanalytikere gøre følgende for at reducere risikoen:

- Reducere produktrisici ved at teste med veldesignede testcases, der entydigt viser om testobjekterne består eller ikke består testen, og ved at deltage i reviews af krav, design og brugerdokumentation
- Implementere passende, risiko-reducerende aktiviteter, der er identificeret i teststrategien og testplanen
- Genvurdere kendte risici baseret på yderligere information indsamlet efterhånden som projektet skrider fremad, og justere sandsynlighed og virkning eller begge dele på passende vis
- Inddrage nye risici, der er fundet ved hjælp af testen.

Test er en måde at reducere produktrisiko på. Når testere finder defekter, reducerer de risikoen ved at defekterne kan rettes før release. Også hvis testerne ikke finder nogle defekter, reducerer test risikoen - den giver vished om, at systemet virker korrekt under de betingelser, der er testet under. Testanalytikere hjælper med at bestemme mulighederne for reduktion af risikoen ved at indsamle præcise testdata, skabe realistiske brugerscenarier og gennemføre eller deltage i undersøgelser af brugervenlighed.

2.4.4.1 Prioritering af tests

Risikoniveaet bruges også til at prioritere tests. En testanalytiker kan fastslå, at der er en høj risiko på området transaktionspræcision i et regnskabssystem. Som et resultat af dette og for at reducere risikoen kan testeren arbejde sammen med andre forretningsdomæneeksperter for at samle et stærkt sæt af stikprøvedata, der kan bearbejdes og kontrolleres for præcision. På samme måde kan en testanalytiker fastslå, at brugervenlighed udgør en væsentlig risiko for et nyt produkt. Snarere end at vente på at en brugeraccepttest finder problemer, kan testanalytikeren prioritere at en tidlig

brugervenlighedstest finder sted på integrationsniveauet for at hjælpe med at finde og løse problemer med brugervenlighed tidligt i testen. Denne prioritering skal overvejes så tidligt som muligt i planlægningsfasen, sådan at tidsplanen kan rumme den nødvendige test på det nødvendige tidspunkt.

I nogle tilfælde afvikles alle test for de højeste risici før eventuelle lavrisikotests, og tests afvikles i streng risikorækkefølge (ofte kaldet "dybde-først"). I andre tilfælde bruges en stikprøvetilgang for at vælge et udpluk af tests på tværs af alle fundne risici, idet risikoen bruges til at vægte udvælgelsen, samtidig med at den sikrer at enhver risiko dækkes mindst en gang (ofte kaldet "bredde-først").

Uanset om risikobaseret test afvikles dybde-først eller bredde-først, er det muligt at den tid, der er sat af til test, løber ud uden at alle tests er blevet afviklet. Risikobaseret test giver testere mulighed for at rapportere til ledelsen i form af udestående risikoniveau på dette tidspunkt, og giver ledelsen mulighed for at beslutte om testen skal udvides eller om de udestående risici skal overføres til brugere, kunder, helpdesk/teknisk support, og/eller driftspersonale.

2.4.4.2 Tilpasning af test til fremtidige testcykler

Risikovurdering er ikke en engangsaktivitet, der gennemføres før starten af testimplementering. Det er en kontinuert proces. Hver fremtidig planlagt testcyklus bør underkastes en ny risikoanalyse, så faktorer som de følgende kan tages med i overvejelserne:

- Alle nye eller væsentligt ændrede produktrisici
- Ustabile eller defekt-tunge områder opdaget under testen
- Risici fra rettede defekter
- Typiske defekter fundet under test
- Områder, der er blevet under-testet (lav testdækning).

Hvis der er afsat ekstra tid til test, kan det være muligt at udvide risikodækningen til områder med lavere risiko.

3 Testteknikker - 825 mins.

Nøgleord

Grænseværdianalyse, årsags-virkningstest, checklistebaseret test, klassifikationstræmetode, kombinatorisk test, beslutningstabeltest, defekttaksonomi, fejlbaseret teknik, domæneanalyse, fejlgætning, ækvivalenspartitionering, erfaringsbaseret teknik, udforskende test, ortogonal tabel, ortogonal tabeltest, parvis test, kravbaseret test, specifikationsbaseret teknik, tilstandsovergangstest, testcharter, usecase test, user story test

Læringsmål for testteknikker

3.2 Specifikationsbaserede teknikker

TA-3.2.1 (K2) Forklar brugen af årsags-virkningsgrafer.
TA-3.2.2 (K3) Skriv testcases ud fra en given specifikation ved brug af testdesignteknikken ækvivalenspartitionering for at opnå en bestemt dækningsgrad.
TA-3.2.3 (K3) Skriv testcases ud fra en given specifikation ved brug af testdesignteknikken grænseværdianalyse for at opnå en bestemt dækningsgrad.
TA-3.2.4 (K3) Skriv testcases ud fra en given specifikation ved brug af testdesignteknikken beslutningstabeller for at opnå en bestemt dækningsgrad.
TA-3.2.5 (K3) Skriv testcases ud fra en given specifikation ved brug af testdesignteknikken tilstandsovergange for at opnå en bestemt dækningsgrad.
TA-3.2.6 (K3) Skriv testcases ud fra en given specifikation ved brug af testdesignteknikken parvis test for at opnå en bestemt dækningsgrad.
TA-3.2.7 (K3) Skriv testcases ud fra en given specifikation ved brug af testdesignteknikken klassifikationstræer for at opnå en bestemt dækningsgrad.
TA-3.2.8 (K3) Skriv testcases ud fra en given specifikation ved brug af testdesignteknikken usecases for at opnå en bestemt dækningsgrad.
TA-3.2.9 (K2) Forklar hvordan user stories bruges som grundlag for testen i et agilt projekt.
TA-3.2.10 (K3) Skriv testcases ud fra en given specifikation ved brug af testdesignteknikken domæneanalyse for at opnå en bestemt dækningsgrad.
TA-3.2.11 (K4) Analyser et system eller dets kravspecifikation for at bestemme, hvilke typer af defekter, man kan forvente at finde, og vælg passende specifikationsbaserede teknik(ker).

3.3 Fejlbaserede teknikker

TA-3.3.1 (K2) Beskriv anvendelsen af fejlbaserede testteknikker og forskellene i deres brug i forhold til specifikationsbaserede teknikker.
TA-3.3.2 (K4) Analyser en given defekttaksonomi for anvendelighed i en bestemt situation ved hjælp af kriterier for velegnet klassifikation.

3.4 Erfaringsbaserede teknikker

TA-3.4.1 (K2) Forklar principperne for erfaringsbaserede teknikker og fordele og ulemper sammenlignet med specifikationsbaserede og fejlbaserede teknikker.
TA-3.4.2 (K3) Specificer udforskende test for et givet scenarie og forklar hvordan resultaterne kan rapporteres.
TA-3.4.3 (K4) Vurdér hvilke specifikationsbaserede, fejlbaserede eller erfaringsbaserede teknikker, der bør anvendes for at opnå specifikke mål i en givet situation i et projekt.

3.1 Introduktion

De testdesignteknikker, der er behandlet i dette kapitel er inddelt i følgende kategorier:
- Specifikationsbaserede (eller adfærdsbaseret eller black-box)
- Fejlbaserede
- Erfaringsbaserede.

Disse teknikker supplerer hinanden og kan bruges efter behov i enhver testaktivitet, uanset på hvilket testniveau den bliver gennemført.

Bemærk, at alle tre kategorier af teknikker kan anvendes til at teste både funktionelle og ikke-funktionelle kvalitetskarateristikker. Test af ikke-funktionelle karakteristikker behandles i næste kapitel.

De testdesignteknikker, der behandles i disse afsnit, fokuserer primært på bestemmelse af optimale testdata (f. eks. ækvivalenspartioner) eller på at udlede testsekvenser (f. eks. tilstandsmodeller). Det er almindeligt at kombinere teknikker for at oprette komplette testcases.

3.2 Specifikationsbaserede teknikker

Specifikationsbaserede teknikker anvendes på testbetingelserne for at udlede testcases baseret på en analyse af testgrundlaget for en komponent eller et system uden reference til dets interne struktur.

Fællestræk for specifikationsbaserede teknikker:
- Modeller, f. eks. oprettes tilstandsovergangsdiagrammer og beslutningstabeller under testdesign i henhold til testteknikken
- Testbetingelser udledes systematisk fra disse modeller.

Nogle teknikker giver også dækningskriterier, som kan bruges til at måle testdesign- og testafviklingsaktiviteter. Fuldstændig opfyldelse af dækningskriterierne betyder ikke, at det fulde sæt af tests er komplet, men kun at modellen ikke længere forslår yderligere tests for at øge dækningen baseret på denne teknik.

Specifikationsbaserede tests er normalt baseret på systemkravdokumenterne. Da kravspecifikationen bør specificere, hvordan systemet skal opføre sig, specielt hvad angår funktionalitet, er udledningen af tests fra kravene ofte en del af testen af systemets adfærd. I nogle tilfælde kan det være, at der ikke er dokumenterede krav, men at krav er underforstået, som f.eks. erstatning af funktionalitet i et ældre system.

Der er et antal specifikationsbaserede testteknikker. Disse teknikker er rettet mod forskellige typer af software og scenarier. Nedenstående afsnit viser anvendeligheden af hver teknik, nogle begrænsninger og problemer, som testanalytikeren kan opleve, måden testdækning måles på og de typer af defekter, der angribes.

3.2.1 Ækvivalenspartitionering

Ækvivalenspartitionering (ÆP) bruges til at nedsætte antallet af testcases, der er nødvendige til effektiv test af håndteringen af inputs, outputs, interne værdier og tidsrelaterede værdier. Partitionering bruges til at oprette ækvivalensklasser (ofte kaldet ækvivalenspartitioner), som er dannet af sæt af værdier, der behandles på samme måde. Ved at vælge en repræsentativ værdi fra en partition, antages dækning for alle elementerne i den samme partition.

Anvendelighed
Denne teknik kan anvendes på alle testniveauer og er hensigtsmæssig, når alle medlemmerne i et sæt af værdier, der skal testes, forventes at blive behandlet på samme måde, og hvor værdisættene anvendt af systemet ikke interagerer. Valget af værdisættene gælder for både gyldige og ugyldige partitioner (dvs. partitioner der indeholder værdier, der bør anses for at være ugyldige for softwaren under test). Denne teknik er stærkest når den bruges i kombination med grænseværdianalyse, der udvider testværdierne til at omfatte dem på kanterne af partitionerne. Dette er en almindelig anvendt teknik for smoke-test af en ny build eller en ny release, da den hurtigt kan bestemme, om den grundlæggende funktionalitet virker.

Begrænsninger/problemer
Hvis antagelsen er ukorrekt og værdierne i partitionen ikke behandles på nøjagtig samme måde, kan denne teknik overse defekter. Det er også vigtigt at vælge partitionerne omhyggeligt. For eksempel, ville et felt, der accepterer positive og negative tal blive bedre testet som to gyldige partitioner, en for de positive tal og en for de negative tal, på grund af sandsynligheden for forskellig behandling. Afhængig af, om nul er tilladt eller ej, kunne dette også blive en partition. Det er vigtigt for testanalytikeren at forstå den underliggende behandling for at kunne bestemme den bedste partitionering af værdier.

Dækning
Dækning bestemmes ved at tage antallet af partitioner, for hvilke en værdi er blevet testet, og dividere med antallet af identificerede partitioner. Brug af flere værdier fra en enkelt partition øger ikke dækningsprocenten.

Defekttyper
Denne teknik finder funktionelle defekter i håndteringen af forskellige dataværdier.

3.2.2 Grænseværdianalyse

Grænseværdianalyse (GVA) bruges til at teste de værdier, der eksisterer på grænserne af ordnede ækvivalenspartitioner. Der er to tilgange til GVA: to-værdi- eller tre-værditest. I to-værditest bruges grænseværdien (på grænsen) og den værdi, der er lige over grænsen (med den mindst mulige forøgelsesenhed). For eksempel, hvis partitionen omfatter værdierne 1 til 10 i forøgelsesenheder på 0,5, vil de to to-værditestværdier for den øvre grænse være 10 og 10,5. Den nederste grænsetests værdier ville være 1 og 0,5. Grænserne er defineret af maksimum- og minimumværdierne i den definerede ækvivalenspartition.

For tre-værditest anvendes værdierne før, på og over grænsen. I det tidligere eksempel ville den øvre grænsetest omfatte 9,5, 10 og 10,5. Den nedre grænsetests ville omfatte 1,5, 1 og 0,5. Beslutningen om der skal anvendes to eller tre grænseværdier bør være baseret på risikoen knyttet til testobjektet, hvor tre-værdi tilgangen anvendt til højrisikoobjekterne.

Anvendelighed
Denne teknik er anvendelig for alle testniveauer og er passende, når der eksisterer ordnede ækvivalenspartitioner. Ordning er nødvendig pga. konceptet om at være på og over grænsen. For eksempel er et interval af tal en ordnet partition. En partition, der består af alle rektangulære objekter, er ikke en ordnet partition og har ikke grænseværdier. Ud over talintervaller kan grænseværdianalyse anvendes på:

- Numeriske attributter af ikke-numeriske variable (f.eks. længde)
- Loops, inklusive dem, der findes i usecases
- Struktur for lagrede data
- Fysiske objekter (inklusive memory)

- Tidsbestemte aktiviteter.

Begrænsninger/problemer
Fordi præcisionen i denne teknik afhænger af at ækvivalenspartitionerne identificeres præcist har grænseværdianalyse de samme begrænsninger og problemer som ækvivalenspartitionering. Testanalytikeren bør også være opmærksom på forøgelsesenhederne i de gyldige og ugyldige værdier for præcist at kunne bestemme de værdier, der skal testes. Kun ordnede partitioner kan bruges i grænseværdianalyse, men denne er ikke begrænset til et interval af gyldige inputs. For eksempel, når der testes for antallet af celler, der er understøttet af et regneark, er der en partition, der indeholder antallet af celler op til og inklusive det maksimale antal tilladte celler (grænsen), og en anden partition, der begynder med en celle over maksimum (over grænsen).

Dækning
Dækning bestemmes ved at tage antallet af grænseværdier, der er testet og dividere det med antallet af identificerede grænseværdier (idet enten to-værdi eller tre-værdi metoden bruges). Dette vil give dækningsprocenten for grænseværditesten.

Defekttyper
Grænseværdianalyse er pålidelig til at finde fejlplacering eller udeladelse af grænser, og kan finde ekstra grænser. Denne teknik finder defekter mht. håndteringen af grænseværdierne, specielt fejl med mindre-end og større-end logik (dvs. forskydning). Den kan også anvendes til at finde ikke-funktionelle defekter, for eksempel tolerance af belastningsgrænser (f. eks. systemet kan klare 10.000 samtidige brugere).

3.2.3 Beslutningstabeller

Beslutningstabeller bruges til at teste interaktion mellem kombinationer af betingelser. Beslutningstabeller giver en klar metode til at verifikationsteste alle betydende kombinationer af betingelser og til at verificere at alle mulige kombinationer er håndteret af softwaren under test. Målet med en beslutningstabeltest er at sikre at enhver kombination af betingelser, relationer og begrænsninger bliver testet. Når man forsøger at teste enhver mulig kombination, kan beslutningstabeller blive meget store. En intelligent måde at reducere antallet af kombinationer fra alle mulige til dem, der er "interessante" kaldes modificeret beslutningstabeltest. Når denne teknik anvendes, reduceres kombinationerne til dem, der vil give forskellige resultater, idet man fjerner de sæt af betingelser, der ikke er relevante for resultatet. Man må slette redundante tests og tests, hvor kombinationen af betingelser ikke er mulig. Beslutningen om at bruge komplette beslutningstabeller eller modificerede beslutningstabeller er normalt risikobaseret. [Copeland03]

Anvendelighed
Denne teknik bruges ofte på integrations-, system- og accepttest-niveau. Afhængig af koden, kan den også anvendes til komponenttest, når en komponent er ansvarlig for et sæt af beslutningslogik. Denne teknik er specielt anvendelig, når kravene er præsenteret i form af flowdiagrammer eller tabeller med forretningsregler. Beslutningstabeller er også en krav-definitionsteknik, og nogle krav kan fremstå direkte i dette format. Selv når kravene ikke præsenteres i tabel- eller flowdiagramform, findes betingelseskombinationerne normalt i den forklarende tekst. Når man designer beslutningstabeller, er det vigtigt at overveje de definerede betingelseskombinationer, så vel som dem, der ikke eksplicit er definerede, men vil eksistere. For at kunne designe en gyldig beslutningstabel skal testerne kunne udlede alle forventede resultater for alle betingelseskombinationer fra specifikationen eller et testorakel. Kun når alle interagerende betingelser er overvejet, kan beslutningstabellen bruges som et godt testdesignværktøj.

Begrænsninger/problemer
Det kan være en udfordring at finde alle de interagerende betingelser, specielt når kravene ikke er veldefinerede eller ikke eksisterer. Det er ikke ualmindeligt at forberede et sæt af betingelser og så finde ud af at det forventede resultat er ukendt.

Dækning
Minimums testdækning for en beslutningstabel er at have en testcase for hver kolonne. Dette antager, at der ikke er sammensatte betingelser og at alle mulige betingelseskombinationer er blevet registreret i en kolonne. Når tests bestemmes ud fra en beslutningstabel, er det også vigtigt at overveje om der er nogle grænsebetingelser, der burde testes. Disse grænsebetingelser kan resultere i, at et højere antal testcases er nødvendigt for at teste software tilstrækkeligt. Grænseværdianalyse og ækvivalenspartitionering supplerer beslutningstabelteknikken.

Defekttyper
Typiske defekter omfatter ukorrekt behandling af specifikke kombinationer af betingelser, som derfor giver uventede resultater. Under udarbejdelsen af beslutningstabellerne, kan der findes defekter i specifikationsdokumentet. De mest almindelige typer af defekter er udeladelser (der er ingen information om, hvad der burde ske i en bestemt situation) og modsigelser. Test kan også finde problemer med betingelseskombinationer, der ikke er håndteret eller ikke håndteret godt.

3.2.4 Årsags-virkningstest

Årsags-virkningsgrafer kan genereres fra en hvilken som helst kilde, der beskriver den funktionelle logik (dvs. "reglerne") for et program, som f.eks. user stories eller flow charts. De kan være gode til at skabe et grafisk overblik over et programs logiske struktur og bliver typisk brugt som grundlag for oprettelse af beslutningstabeller. Fastholdelse af beslutninger i årsags-virkningsgrafer og/eller beslutningstabeller muliggør opnåelse af systematisk testdækning af programmets logik.

Anvendelighed
Årsags-virkningsgrafer kan anvendes i de samme situationer som beslutningstabeller og også til de samme testniveauer. Specielt viser en årsags-virkningsgraf betingelseskombinationer, der giver resultater (kausalitet), betingelseskombinationer, der udelukker resultater (ikke), multiple betingelser, der skal være sande for at give et resultat (og), og alternative betingelser, der kan være sande for at give et specifikt resultat (eller). Disse relationer kan være lettere at se i en årsags-virkningsgraf end i tekstuel beskrivelse.

Begrænsninger/problemer
Årsags-virkningstest kræver ekstra tid og indsats at lære sammenlignet med andre testdesignteknikker. Det kræver også værktøjsunderstøttelse. Årsags-virkningsgrafer har en speciel notation, der skal forstås af designeren og den, der læser grafen.

Dækning
Enhver mulig årsag-til-effekt linje skal tests, inkl. kombinationsbetingelserne, for at opnå minimum dækning. Årsags-virkningsgrafer omfatter en måde at definere begrænsninger vedr. data og begrænsninger vedr. det logiske flow på.

Defekttyper
Disse grafer finder de samme typer kombinatoriske defekter som findes med beslutningstabeller. Ud over det hjælper oprettelsen af graferne med at definere det nødvendige detaljeringsniveau i testgrundlaget, og hjælper således med at forbedre detaljegraden og kvaliteten af testgrundlaget samt hjælper testeren med at opdage manglende krav.

3.2.5 Tilstandsovergangstest

Tilstandsovergangtest bruges til at teste, hvordan softwaren er i stand til at gå ind i og ud af definerede tilstande via gyldige og ugyldige overgange. Hændelser får softwaren til at overgå fra tilstand til tilstand og udføre aktioner. Hændelser kan være forsynet med betingelser (nogle gange kaldet bevogtningsbetingelser eller overgangsbevogtninger), som har indflydelse på, hvilken overgangssti der skal følges. For eksempel vil en login-hændelse med en gyldig brugernavn/password-kombination resultere i en anden overgang end en login-hændelse med et ugyldigt password.

Tilstandsovergange følges enten i et tilstandsovergangsdiagram, der viser alle de gyldige overgange mellem tilstande i et grafisk format, eller i en tilstandstabel, der viser alle potentielle overgange, både gyldige og ugyldige.

Anvendelighed
Tilstandsovergangstest kan anvendes for al software, der har definerede tilstande og har hændelser, der skaber overgang mellem disse tilstande (f.eks. skift af skærmbillede). Tilstandsovergangstest kan anvendes på alle testniveauer. Indlejret software, web software, og alle typer af transaktionssoftware er gode kandidater til denne type af test. Kontrolsystemer, dvs. trafiklyskontroller, er også gode kandidater til denne type af test.

Begrænsninger/problemer
Det at bestemme tilstandene er ofte den sværeste del af at definere tilstandstabellen eller diagrammet. Når softwaren har en brugergrænseflade bliver de forskellige skærmbilleder, der vises for brugeren, ofte anvendt til at definere tilstandene. For indlejret software kan tilstandene være afhængige af de tilstande, som hardwaren vil opleve.

Bortset fra selve tilstandene er den mindste enhed i tilstandsovergangtest den enkelte overgang, også kendt som en 0-switch. Bare det at teste alle overgange vil finde nogle typer af tilstandsovergangsdefekter, men flere kan findes ved at teste sekvenser af overgange. En sekvens af to sammenhængende overgange kaldes en 1-switch, en sekvens af 3 sammenhængende overgange er en 2-switch, og så videre. (Disse switches betegnes somme tider også som N-1 switches, hvor N repræsenterer antallet af overgange, der vil blive gennemløbet. En enkelt overgang for eksempel (en 0-switch) ville være en 1-1 switch. [Bath08]

Dækning
Som for andre typer af testteknikker er der et hierarki af testdækning. Den mindst acceptable grad af dækning er at have besøgt alle tilstande og gennemløbet alle overgange. 100 % overgangsdækning (også kendt som 100 % 0-switch dækning eller 100 % logisk forgreningsdækning) vil garantere at alle tilstande er besøgt og alle overgange gennemløbet, med mindre systemdesignet eller tilstandsovergangsmodellem (diagram eller tabel) er fejlbehæftede. Afhængig af relationerne mellem tilstande og overgange, kan det være nødvendigt at gennemløbe nogle overgange mere end en gang for at afvikle andre overgange en enkelt gang.

Begrebet "n-switch dækning" er relateret til antallet af dækkede overgange. For eksempel kræver opnåelse af 100 % 1-switch dækning, at alle gyldige sekvenser af to sammenhængende overgange er blevet testet mindst en gang. Denne test kan stimulere andre typer af afvigelser, som 100 % 0-switch dækning ville "overse".

"Round-trip-dækning" anvendes i situationer, hvor sekvenser af overgange giver loops. 100 % round-trip-dækning er opnået, når alle loops fra alle tilstande tilbage til den samme tilstand er blevet testet. Dette skal testes for alle tilstande, der er med i loops.

For enhver af disse tilgange vil en stadig højere grad af dækning forsøge at omfatte alle ugyldige overgange. Dækningskrav og dækningssæt for tilstandsovergangstest skal beskrive om ugyldige overgange er omfattet eller ej.

Defekttyper
Typiske defekter omfatter ukorrekt behandling i den aktuelle tilstand som et resultat af den behandling, der fandt sted i den tidligere tilstand, ukorrekte eller ikke-understøttede overgange, tilstande uden exits og behovet for tilstande eller overgange, der ikke findes. Under oprettesen af tilstandsmaskinemodellen kan der findes defekter i specifikationsdokumentet. De mest almindelige typer af defekter er udeladelser (der er ingen information om, hvad der faktisk skal ske i en bestemt situation) og modsigelser.

3.2.6 Kombinatoriske testteknikker

Kombinatorisk test anvendes, når der testes software med adskillige parametre, hvor hver af disse har adskillige værdier, hvilket giver anledning til flere kombinationer, end det er muligt at teste inden for den givne tid. Parametrene skal være uafhængige og kompatible forstået på den måde, at enhver mulighed for enhver faktor kan kombineres med enhver mulighed for enhver af de andre faktorer. Klassifikationstræer tillader at sådanne kombinationer udelukkes, hvis visse muligheder ikke er kompatible. Det kan ikke udelukkes at de kombinerede faktorer ikke påvirker hinanden - det kan de meget vel, men de bør påvirke hinanden på en måde som er tilladt fra et forretnings- eller teknisk perspektiv.

Kombinatorisk test giver en mulighed for at bestemme passende delmængder af disse kombinationer for at opnå et forudbestemt niveau af dækning. Antallet af elementer, der skal være i kombinationerne kan vælges af testanalytikeren, inklusive enkeltelementer, par, tre eller flere [Copeland03]. Der findes værktøjer, der kan hjælpe testanalytikeren med denne opgave (se www.parvis.org for eksempler). Disse værktøjer kræver enten at parametrene og deres værdier oplistes (parvis test og ortogonal tabel test) eller bliver præsenteret i grafisk form (klassifikationstræer) [Grochtmann94]. Parvis test er en metode, der anvendes til at teste par af variable i kombination. Ortogonale tabeller er prædefinerede, matematisk korrekte tabeller, der tillader testanalytikeren at udskifte de elementer, der skal testes med variabler i et array og dermed danne et sæt af kombinationer, der vil opnå et dækningsniveau når de bliver testet [Koomen06]. Klassifikationstræsværktøjer tillader testanalytikeren at bestemme størrelsen af kombinationer, der skal testes (dvs. kombinationer af to værdier, tre værdier osv.).

Anvendelighed
Problemet med mange kombinationer af parameterværdier viser sig blandt andet i disse situationer:

- Nogle testcases indeholder mange parametre, som hver har et stort antal mulige værdier, for eksempel en skærm med mange input-felter. Her er det forskellige kombinationer af parameterværdier, der er input til testcases
- Nogle systemer kan konfigures i mange dimensioner, og det giver rigtig mange varianter.

I begge disse situationer kan man anvende kombinatorisk test til at bestemme en delmængde af kombinationer af en rimelig størrelse.

For parametre med mange værdier kan man f.eks. bruge ækvivalenspartitionering til at udvælge et antal værdier for hver parameter. Derefter kan man bruge kombinatorisk test til at reducere antallet af kombinationer. Disse teknikker anvendes normalt anvendt til integrations-, system- og systemintegrationstest-niveauerne.

Begrænsninger/problemer
Den største begrænsning for disse teknikker er antagelsen om, at resultaterne fra nogle få tests er repræsentative for alle tests, og at disse få tests repræsenterer den forventede brug. Hvis der er en uventet interaktion mellem bestemte variable, risikerer man at den sammenhæng ikke bliver testet. Teknikken med reduktion i antallet af tests kan være vanskelig at forklare for et ikke-teknisk publikum.

Det kan være vanskeligt at bestemme parametrene og deres værdier, og det er en svær opgave at finde et begrænset antal kombinationer, der opfylder et bestemt niveau for dækning. Derfor anvender man normalt et værktøj til at bestemme minimumssættet af kombinationer. Nogle værktøjer understøtter muligheden for at tvinge nogle (del-) kombinationer ind i eller ud af det endelige udvalg af kombinationer. Testanalytikeren kan benytte denne mulighed til at fremhæve eller nedtone faktorer baseret på domænekendskab eller information om brugen af produktet.

Dækning
Der er adskillige niveauer af dækning. Det laveste niveau kaldes 1-vis eller enkelt dækning. Det kræver, at hver værdi af hver parameter er til stede i mindst en af de valgte kombinationer. Det næste niveau af dækning kaldes 2-vis eller parvis dækning. Det kræver, at hvert par af værdier for hver to parametre er til stede i mindst en kombination. Denne ide kan generaliseres til n-vis dækning, hvilket kræver at enhver del-kombination af værdier af ethvert sæt af n parametre er til stede i det valgte sæt af kombinationer. Jo større n, jo flere kombinationer er nødvendige for at opnå 100 % dækning. Minimum dækning med disse teknikker er at have en testcase for hver kombination, der fremstilles af værktøjet.

Defekttyper
Den mest almindelige type af defekter, der findes med denne type af test, er defekter relateret til de kombinerede værdier af adskillige parametre.

3.2.7 Usecase test

Usecase test giver transaktions- og scenariebaserede tests, der bør efterligne brugen af systemet. Usecases er defineret i form af interaktioner mellem aktører og systemet, der opfylder et eller andet mål. Aktører kan være brugere eller eksterne systemer.

Anvendelighed
Usecase test anvendes normalt på system- og accepttestniveauerne. Den kan anvendes til integrationstest afhængig af integrationsniveauet, og endda til komponenttest afhængig af komponentens adfærd. Usecases danner også ofte grundlag for performancetest, fordi de portrætterer realistisk brug af systemet. Scenarierne beskrevet i usecases kan fordeles til virtuelle brugere for at skabe en realistisk belastning på systemet.

Begrænsninger/problemer
For at være gyldige skal usecases gengive realistiske handlinger. Denne information bør komme fra en bruger eller en brugerrepræsentant. Værdien af en usecase er mindre, hvis den ikke præcist gengiver en rigtig brugers handlinger. En præcis definition af de forskellige alternative stier (flows) er vigtigt for at testdækningen kan blive grundig. Usecases bør tages som en retningslinje, men ikke en komplet definition af, hvad der bør testes, da de måske ikke giver en klar definition af det fulde sæt af krav. Det kan også være værdifuldt at fremstille andre modeller som f.eks. flow-diagrammer ud fra teksten fra usecases for at gøre testen mere præcis og for at verificere selve usecasen.

Dækning
Minimum dækning for en usecase er én testcase for den positive hovedsti og én testcase for hver alternativ sti. Alternative stier består af undtagelser og fejl. De vises nogle gange som udvidelser af

hovedstien. Dækningsprocenten for usecases beregnes ved at dividere antallet af testede stier med det totale antal hovedstier og alternative stier.

Defekttyper
Defekter omfatter forkert behandling af definerede scenarier, manglende behandling af alternative stier, ukorrekt behandling af de givne betingelser og ubehjælpsom eller ukorrekt rapportering.

3.2.8 User story-test

I nogle agile metoder, som f.eks. Scrum, forberedes kravene i form af user stories, som beskriver små funktionelle enheder, der kan designes, udvikles, testes og demonstreres i en enkelt iteration [Cohn04]. Disse user stories omfatter en beskrivelse af den funktionalitet, der skal implementeres, eventuelle ikke-funktionelle kriterier, og omfatter også acceptkriterier, der skal opfyldes for at en user story kan opfattes om gennemført.

Anvendelighed
User stories er anvendes fortrinsvis i agile og tilsvarende iterative og inkrementelle miljøer. De bruges både til funktionel test og ikke-funktionel test. User stories bruges til test på alle niveauer med den forventning, at udvikleren vil demonstrere user storyens implementerede funktionalitet før koden overlades til teammedlemmerne med de næste niveauer af testopgaver (f.eks. integration, performance test).

Begrænsninger/problemer
Da stories er små forøgelser af funktionalitet, kan der være krav til at fremstille drivere og stubbe for at kunne teste den funktionalitetsbid, der er leveret. Dette kræver som regel en mulighed for at programmere og bruge værktøjer, der kan understøtte testen, som f.eks. API testværktøjer. Oprettelse af driverne og stubbene er normal udviklerens ansvar, selv om en teknisk testanalytiker også kan være involveret i at fremstille denne kode og bruge API testværktøjerne. Hvis der anvendes en kontinuert integrationsmodel, som det er tilfældet i de fleste agile projekter, er behovet for drivere og stubbe minimaliseret.

Dækning
Minimum dækning af en user story er at verificere, at hvert af de specificerede acceptkriterier er blevet opfyldt.

Defekttyper
Defekter er normalt funktionelle på den måde, at softwaren ikke leverer den specificerede funktionalitet. Defekter i form af integrationsproblemer med funktionaliteten i den nye story med den funktionalitet, der allerede findes, ses også. Fordi stories kan være udviklet uafhængigt, kan der forekomme performance-, interface- og fejlhåndteringsproblemer. Det er vigtigt for testanalytikeren at teste både den leverede funktionalitet og integrationsteste hver gang en ny story frigives til test.

3.2.9 Domæneanalyse

Et domæne er et defineret sæt af værdier. Sættet kan være defineret som et interval af værdier af en enkelt variabel (et endimensionelt domæne, f.eks. "mænd over 24 år og under 66 år"), eller som intervaller af værdier af interagerende variable (et multi-dimensionelt domæne, f. eks., "mænd over 24 år og under 66 år og med vægt over 69 kg og under 90 kg"). Hver testcase for et multi-dimensionelt domæne skal omfatte passende værdier for alle variable.

Domæneanalyse af et endimensionelt domæne anvender typisk ækvivalenspartitionering og grænseværdianalyse. Når partitionerne er defineret, vælger testanalytikeren værdier fra hver partition, der repræsenterer en værdi, der er i partitionen (INDE), uden for partitionen (UDE), på grænsen af

partitionen (PÅ) og lige uden for partitionens grænse (UDENFOR). Ved bestemmelse af disse værdier testes hver partition sammen med dens grænsebetingelser. [Black07]

Brug af ækvivalenspartitionering og grænseværdianalyse vil få antallet af testcases til at stige eksponentielt. I stedet kan man bruge domæneteori, der kun vil give en lineær stigning i antal testcases. Desuden vil domæne-metoden kunne finde defekter i multi-dimensionelle domæner, som de større testsæt ikke vil finde. Det er fordi domæne-metoden indeholder en defektteori (en fejlmodel), som ækvivalenspartitionering og grænseværdianalyse mangler. For multi-dimensionelle domæner kan testmodellen være konstrueret som en beslutningstabel (eller "domænematrice"). Bemærk, at det kræver computerberegninger at bestemme testcase-værdier for domæner med tre eller flere dimensioner.

Anvendelighed
Domæneanalyse kombinerer de teknikker, der anvendes for beslutningstabeller, ækvivalenspartitionering og grænseværdianalyse til at danne et mindre sæt af tests, der stadig dækker de vigtigste områder og de sandsynlige fejlområder. Den anvendes ofte, hvor beslutningstabeller ville være uhåndterlige pga. et stort antal potentielt interagerende variable. Domæneanalyse kan foretages på ethvert testniveau, men er mest brugt på integrations- og systemtestniveau.

Begrænsninger/problemer
Udførelse af grundig domæneanalyse kræver en stærk forståelse af softwaren for at identificere de forskellige domæner og mulig interaktion mellem domænerne. Hvis et domæne er uidentificeret, kan testen være mangelfuld. Det er dog sandsynligt at domænet vil blive opdaget, for UDENFOR og UDE-variablerne vil måske lande i det uopdagede domæne. Domæneanalyse er en stærk teknik at bruge, når man arbejder sammen med en udvikler for at afgrænse en test.

Dækning
Minimumsdækning for en domæneanalyse vil være at have en test for hver INDE, UDE, PÅ og UDENFOR-værdi for hvert domæne. Hvor der er et overlap mellem værdierne (for eksempel når en UDE-værdi for ét domæne altid er en INDE-værdi for et andet domæne), er der ingen grund til at gentage testen. Derfor er der ofte færre end fire tests pr. domæne.

Defekttyper
Defekter omfatter funktionelle problemer inden for domænet, grænseværdihåndtering, problemer med interaktion mellem variable og fejlhåndtering (specielt for de værdier, der ikke er i et gyldigt domæne).

3.2.10 Kombination af teknikker

Nogle gange kombineres teknikker for at oprette testcases. For eksempel kan man bruge ækvivalenspartitionering på de betingelser, som man har identificeres ved hjælp af en beslutningstabel til at finde flere måder hvorpå en betingelse kan opfyldes. Testcases ville på den måde dække ikke alene enhver kombination af betingelser, men også, for de betingelser, der blev delt, ville yderligere testcases kunne oprettes for at dække ækvivalenspartitioner. Når den specifikke test til anvendelse vælges, skal testanalytikeren overveje anvendelsen af teknikken, begrænsningerne og problemerne og testens mål i form af dækning og defekter, der skal findes. Der er måske ikke en enkelt ”bedste” teknik for en situation. Kombinerede teknikker vil ofte give en mest komplette dækning, hvis der er tilstrækkeligt med tid og evner til det.

3.3 Fejlbaserede teknikker

3.3.1 Anvendelse af fejlbaserede teknikker

En fejlbaseret testdesignteknik er en, hvor typen af den søgte defekt bruges som grundlag for testdesign, hvor tests udledes systematisk ud fra, hvad der vides om defekttypen. I modsætning til specifikationsbaseret test, som udleder sine tests fra specifikationen, udleder fejlbaseret test tests ud fra defekttaksonomier (dvs. kategoriserede lister), der kan være fuldstændig uafhængige af den software, der skal testes. Taksonomier kan omfatte lister af defekttyper, underliggende årsager, fejlsymptomer og andre defekt-relaterede data. Fejlbaseret test kan også anvende lister af fundne risici og risikoscenarier som grundlag for målrettet test. Denne testteknik giver testerne mulighed for at angribe en specifik fejltype eller for at arbejde systematisk gennem et defekttaksonomi af kendte og almindelige fejl inden for en specifik type. Testanalytikeren anvender taksonomidata til at bestemme formålet med testen, som er at finde en specifik type af defekt. Ud fra denne information vil testanalytikeren opstille testcasene og testbetingelser, der vil få defekten til at vise sig, hvis den findes.

Anvendelighed
Fejlbaseret test kan anvendes på ethvert testniveau, men er mest almindelig anvendt for systemtest. Der findes standardtaksonomier, der passer til mangfoldige typer software. Denne ikke-produktspecifikke type af test hjælper med til at løfte industriens standardkendskab til at udlede de specifikke tests. Ved at holde sig til industri-specifikke taksonomier kan metrikker vedr. forekomst af defekter følges på tværs af projekter og endog på tværs af organisationer.

Begrænsninger/problemer
Der findes mangfoldige defekttaksonomier og de kan være fokuseret på specifikke testtyper, som f.eks. brugervenlighed. Det er vigtigt at vælge en taksonomi, der passer til den software, der skal testes, hvis der er nogen til rådighed. For eksempel kan det være, at der ikke er nogle taksonomier til rådighed for innovativ software. Nogle organisationer har samlet deres egne taksonomier over sandsynlige eller ofte forekommende defekter. Uanset hvilken taksonomi, der anvendes, er det vigtigt at definere den forventede dækning, før testen startes.

Dækning
Teknikken giver dækningskriterier som anvendes til at afgøre, hvornår alle brugbare testcases er identificeret. En særlig ting er, at dækningskriterier for fejlbaserede teknikker har en tendens til at være mindre systematiske end for specifikationsbaserede teknikker, fordi kun generelle regler for dækning er givet og den specifikke beslutning om, hvad der udgør grænsen for anvendelig dækning er skønsmæssig. Som med andre teknikker betyder dækningskriterier ikke, at det fulde sæt af tests er komplet, men snarere at defekter, der overvejes, ikke længere giver anledning til tests baseret på denne teknik.

Defekttyper
De defekttyper, der findes, afhænger normalt af den anvendte taksonomi. Hvis der anvendes en taksonomi for brugergrænseflade, vil størstedelen af de defekter, der findes, sandsynligvis være brugergrænserelaterede, men andre defekter kan findes som et biprodukt af den specifikke test.

3.3.2 Defekttaksonomier

Defekttaksonomier er kategoriserede lister over defekttyper. Disse lister kan være meget generelle og anvendt som retningslinjer på højt niveau, eller de kan være meget specifikke. For eksempel kunne en taksonomi for brugergrænsefladedefekter indeholde generelle elementer som f.eks. funktionalitet, fejlhåndtering, grafisk display og performance. En detaljeret taksonomi kunne omfatte en liste af alle

mulige brugergrænsefladeobjekter (specielt for grafisk brugergrænseflade) og kunne udpege forkert håndtering af disse objekter, som f.eks.:

Tekstfelter:
- Gyldige data accepteres ikke
- Ugyldige data accepteres
- Længden på input verificeres ikke
- Specielle tegn opdages ikke
- Brugerfejlmeddelelser er ikke informative
- Bruger er ikke i stand til at rette forkerte data
- Regler anvendes ikke.

Datofelter:
- Gyldige datoer accepteres ikke
- Ugyldige datoer afvises ikke
- Datointervaller verificeres ikke
- Præcisionsdata behandles ikke korrekt (f.eks. tt:mm:ss)
- Bruger er ikke i stand til at rette fejlagtige data
- Regler anvendes ikke (f.eks. at slutdato skal være større end startdato).

Der er mange defekttaksonomier til rådighed spændende fra formelle taksonomier, der kan købes, til dem, der er beregnet til specifikke formål for forskellige organisationer. Internt udviklede defekttaksonomier kan også anvendes målrettet mod specifikke defekter, der ofte findes i organisationen. Når der oprettes en ny defekttaksonomi eller et, der er til rådighed, tilpasses, er det vigtigt først at definere målet eller formålene med taksonomien. For eksempel kan målet være at undersøge problemer i brugergrænsefladen, der er blevet opdaget i produktionssystemer, eller at undersøge problemer med inputfelter.

Oprettelse af en taksonomi:
1. Sæt et mål og definer det ønskede detaljeringsniveau
2. Vælg en given taksonomi som grundlag
3. Definer værdier og almindelige defekter, der er fundet i organisationen, og/eller som er fundet og anvendes i andre organisationer.

Jo mere detaljeret taksonomien er, jo længere tid vil det tage at udvikle og vedligeholde den, men det vil resultere i at det vil være lettere at genskabe testresultaterne. Detaljerede taksonomier kan være redundante, men de tillader et testteam at opdele testen uden tab af information eller dækning.

Når den passende taksonomi er valgt, kan den bruges til at oprette testbetingelser og testcases. En risikobaseret taksonomi kan hjælpe med at fokusere testen på et specifikt risikoområde. Taksonomier kan også anvendes for ikke-funktionelle områder, som f.eks. brugervenlighed, performance, osv. Taksonomilister er tilgængelige i forskelige publikationer, fra IEEE, og på internettet.

3.4 Erfaringsbaserede teknikker

Erfaringsbaserede tests udnytter testernes kunnen og intuition sammen med deres erfaring med tilsvarende applikationer eller teknologier. Disse tests er effektive til at finde defekter, men ikke så velegnede som andre teknikker til at opnå bestemte testdækningsniveauer eller fremstille genbrugelige testprocedurer. I tilfælde, hvor systemdokumentationen er ringe, testtiden er kraftigt begrænset eller testteamet har stor ekspertise i systemet, der skal testes, kan erfaringsbaseret test være et godt alternativ til mere strukturerede tilgange. Erfaringsbaseret test kan være uegnet for

systemer, der kræver detaljeret testdokumentation, høje niveauer af repeterbarhed eller mulighed for præcist at vurdere testdækning.

Når der anvendes dynamiske og heuristiske tilgange, anvender testere normalt erfaringsbaserede tests, og test er mere reaktive i forhold til hændelser end forudplanlagte testtilgange. Oven i købet er afvikling og vurdering samtidige aktiviteter. Nogle strukturerede tilgange til erfaringsbaserede tests er ikke fuldstændig dynamiske, dvs. at testene ikke oprettes på fuldstændig samme tid som testerene afvikler testen.

Bemærk, at selv om nogle dækningsideer præsenteres for de teknikker, der behandles her, har erfaringsbaserede teknikker ikke formelle dækningskriterier.

3.4.1 Fejlgætning

Når der anvendes fejlgætningsteknik, bruger testanalytikeren erfaring til at gætte de mulige fejltagelser, der kan være blevet begået, da koden blev designet og udviklet. Når man har fundet de forventede fejltagelser, afgør testanalytikeren hvilken metode der er bedst egnet til at afdække de resulterende defekter. Hvis testanalytikeren f.eks. forventer at softwaren vil udvise afvigelser, når man indtaster et ugyldigt password, vil han eller hun designe tests hvor man skal indtaste forskellige værdier i password-feltet. Under testen kan man så verificere om fejltagelsen faktisk er sket og dermed har skabt en defekt, der kan konstateres som en afvigelse, når testene afvikles.

Ud over at blive anvendt som en testteknik, kan fejlgætning også bruges til at finde mulige afvigelsestilstande ved risikoanalyse. [Myers79]

Anvendelighed
Fejlgætning foretages primært under integrations- og systemtest, men kan anvendes på alle niveauer af test. Denne teknik anvendes ofte med andre teknikker og hjælper med at udvide omfanget af de eksisterende testcases. Fejlgætning kan også anvendes effektivt, når en ny release af softwaren skal testes for almindelige fejltagelser og fejl, før en mere streng og beskrevet test begynder. Checklister og taksonomier kan være nyttige til at guide testen.

Begrænsninger/problemer
Dækning er svær at vurdere og variere voldsomt med testanalytikerens evner og erfaring af. Den er bedst anvendt af en erfaren tester, som kender til de typer af defekter, der almindeligvis introduceres i den type af kode, der testes. Fejlgætning er almindelig anvendt, men er ofte ikke dokumenteret og kan derfor være mindre reproducerbar end andre former for test.

Dækning
Når der anvendes en taksonomi, bestemmes dækningen af de tilhørende datafejl og defekttyper. Uden en taksonomi, er dækningen begrænset af testerens erfaring og kunnen og den tid, der er til rådighed. Afkastet fra denne teknik vil variere baseret på, hvor godt testerne kan angribe problematiske områder.

Defekttyper
Typiske defekter er normalt dem, der er defineret i den specifikke taksonomi eller “gættet” af testanalytikeren, der ikke kunne have været fundet i den specifikationsbaserede test.

3.4.2 Checkliste-baseret test

Når den checklistebaserede test teknik anvendes, bruger den erfarne testanalytiker en højniveau, generaliseret liste af elementer, der skal lægges mærke til, kontrolleres eller huskes, eller et sæt af regler eller kriterier mod hvilke et produkt skal verificeres. Disse checklister er opbygget baseret på et

sæt af standarder, erfaring og andre overvejelser. En standard checkliste for en brugergrænseflade, der anvendes som grundlag for test af en applikation, er et eksempel på en checklistebaseret test.

Anvendelighed
Checklistebaseret test anvendes mest effektivt i projekter med et erfarent testteam, der kender softwaren under test eller kender det område, der er dækket af listen (f.eks. for at kunne anvende en brugergrænsefladecheckliste, kan testanalytikeren kende til brugergrænsefladetest, men ikke til den specifikke software under test). Da checklister er på højniveau og har en tendens til at mangle de detaljerede trin, der almindeligvis finder testcases og testprocedurer, bruges testernes kendskab til at fylde hullerne. Ved at fjerne de detaljerede trin kræver checklister mindre vedligeholdelse og kan anvendes til mange ensartede releases. Checklister kan anvendes på alle testniveauer. Checklister anvendes også til regressionstest og smoke-test.

Begrænsninger/problemer
Checklisters højniveau-natur kan påvirke reproducerbarheden af testresultater. Forskellige testere vil muligvis fortolke en checkliste forskelligt og vil måske handle forskelligt for at opfylde listens punkter. Dette kan forårsage forskellige resultater, selv om den samme checkliste blev anvendt. Dette kan give bredere dækning, men gør det svært at gentage testen nøjagtigt. Checklister kan også resultere i overdreven tillid til det dækningsniveau, der er opnået, eftersom den faktiske test afhænger af testernes bedømmelse under udførelsen. Checklister kan udledes fra mere detaljerede testcases eller lister, og har en tendens til at vokse over tid. Det er nødvendigt at vedligeholde checklister for at de til stadighed dækker de vigtigste aspekter af den software, der testes.

Dækning
Dækningen er lige så god som checklisten, men på grund af checklisters højniveau-natur, vil resultaterne variere afhængig af den testanalytiker, der afvikler checklisten.

Defekttyper
Typiske defekter fundet med denne teknik omfatter fejl, der er et resultat af at variere data, rækkefølgen af trin eller det generelle workflow under test. Brug af checklister kan hjælpe med til at holde testen frisk, da nye kombinationer af data og processer er tilladt under test.

3.4.3 Udforskende test

Udforskende test er karakteriseret ved at testerne samtidig lærer om produktet og dets defekter, planlægger det testarbejde, der skal udføres, designer og afvikler testene, og rapporterer resultaterne. Testeren tilpasser testmålene dynamisk under afviklingen og forbereder kun letvægtsdokumentation. [Whittaker09]

Anvendelighed
God udforskende test er planlagt, interaktiv og kreativ. Den kræver ikke meget dokumentation af systemet, der skal testes, og anvendes ofte i situationer, hvor dokumentationen ikke er til rådighed eller ikke er tiltrækkelig for andre testteknikker. Udforskende test anvendes ofte til at supplere andre test og for at danne grundlag for udviklingen af ekstra testcases.

Begrænsninger/problemer
Det kan være vanskeligt at styre og planlægge udforskende test. Det kan være svært at gentage en test, og den samlede dækning kan være tilfældig. Testmanageren kan bruge testcharters til at udpege emnet for en testsession og time-boxing til at bestemme den maksimale varighed. Efter sessionen kan testmanageren afholde debriefing og fastlægge charters for de kommende sessioner. Sådanne møder fungerer bedst i mindre projekter.

Det kan være svært at følge op på udforskende sessioner i et teststyringssystem. Man kan gøre det ved at registrere udforskende sessioner som testcases. På den måde kan man følge op på den planlagte og forbrugte tid og den forventede og opnåede dækning.

Det er vanskeligt at reproducere udforskende test. Det har man f.eks. behov for når man skal demonstrere en afvigelse. Nogle benytter muligheden for capture/playback testautomatiserings-værktøjer til at optage udforskende test. Det kan være en stor opgave efterfølgende at gennemgå detaljerne for at finde en årsag til en bestemt afvigelse, men i det mindste er der så en optagelse af alle trin i testen.

Dækning
Chartre kan oprettes til at specificere opgaver, formål, og leverancer. Herefter kan man planlægge udforskende sessioner for at opnå formålene. Charteret kan også identificere, hvor testarbejdet skal fokuseres, hvad der er inden for og uden for testsessionens omfanget, og hvilke ressourcer der skal være forpligtet til at færdiggøre de planlagte tests. En session kan anvendes til at fokusere på specifikke defekttyper og problematiske områder, der kan testes uden at behøve den beskrevne tests formalisme.

Defekttyper
Typiske defekter fundet ved udforskende test er scenariebaserede problemer, der blev overset under beskrevne funktionelle test, problemer, der falder mellem funktionelle grænser, og workflows-relaterede problemer. Performance og sikkerhedsproblemer findes også somme tider under udforskende test.

3.4.4 Anvendelse af den bedste teknik

Defekt- og erfaringsbaserede teknikker kræver anvendelse af kendskab til defekter og andre testerfaringer for at målrette testen for at forøge defektopdagelsen. Disse strækker sig fra “hurtigtest”, hvor testeren ikke har nogle formelt forudplanlagte aktiviteter at udføre, over forudplanlagte sessioner til beskrevne sessioner. De er næsten altid brugbare, men har specielt værdi under de følgende omstændigheder:

- Der er ikke specifikationer til rådighed
- Der er ringe dokumentation af systemet under test
- Der er afsat utilstrækkelig tid til at designe og oprette detaljerede tests
- Testere har erfaring med domænet og/eller teknologien
- Afvigelse fra beskreven test er et mål for at maksimere testdækning
- Operationelle afvigelser skal analyseres.

Defekt- og erfaringsbaserede teknikker er også gode, når de anvendes sammen med specifikationsbaserede teknikker, da de fylder huller i testdækning, der stammer fra systematiske svagheder i disse teknikker. Som med de specifikationsbaserede teknikker er der ikke én perfekt teknik for alle situationer. Det er vigtigt at testanalytikeren forstår fordele og ulemper ved hver teknik og at han eller hun kan vælge den bedste teknik eller det bedste sæt af teknikker i situationen. Her må han eller hun overveje projekttype, tidsplan, adgang til information, testernes evner og andre faktorer, der kan påvirke valget.

4 Test af kvalitetskarakteristikker - 120 min.

Nøgleord

Adgangstest, nøjagtighedstest, attraktivitet, heuristisk vurdering, tværoperationalitetstest, læringsegnethed, brugsegnethed, egnethedstest, SUMI, forståelsesegnethed, brugervenlighedstest, WAMMI

Læringsmål for test af kvalitetskarakteristikker

4.2 Test af forretningsdomæner

TA-4.2.1 (K2) Forklar med eksempler hvilke testteknikker, der er hensigtsmæssige til test af karakteristikker for nøjagtighed, egnethed, tværoperationalitet og overensstemmelse.

TA-4.2.2 (K2) Beskriv de defekter, der typisk kan findes ved at teste for nøjagtighed, egnethed og tværoperationalitet.

TA-4.2.3 (K2) Beskriv hvornår i livscyklen men bør teste for nøjagtighed, egnethed og tværoperationalitet.

TA-4.2.4 (K4) I et givet projekt: Sammenfat hvordan man kan verificere og validere implementering af brugervenlighedskrav og opfyldelsen af brugernes forventninger.

4.1 Introduktion

Mens de forrige kapitler beskriver specifikke teknikker, der er til rådighed for testerne, behandler dette kapitel anvendelsen af disse teknikker i evalueringen af de vigtigste karakteristikker, der anvendes til at beskrive kvaliteten af softwareapplikationer eller systemer.

Dette pensum behandler de kvalitetskarakteristikker, som kan evalueres af en testanalytiker. De attributter, der skal evalueres af den tekniske testanalytiker er behandlet i pensum for Advanced Technical Test Analyst. Beskrivelsen af produktkvalitetskarakteristikker, der er givet i ISO 9126, er anvendt som retningslinje for beskrivelse af karakteristikkerne. Andre standarder, som f.eks. ISO 25000 [ISO25000] serien (som har overtaget ISO 9126) kan også anvendes. ISO kvalitetskarakteristikkerne er opdelt i produktkvalitetskarakteristikker (attributter), som alle kan have under-karakteristikker (under-attributter). Disse er vist i tabellen neden for sammen med en indikation af, hvilke karakteristikker/under-karakteristikker der dækkes af testanalytiker og teknisk testanalytiker pensaene:

Karakteristik	**Underkarakteristikker**	**Testanalytiker**	**Teknisk testanalytiker**
Funktionalitet	Nøjagtighed, egnethed, tværoperationalitet, overensstemmelse	X	
	Sikkerhed		X
Pålidelighed	Modenhed (robusthed), fejl-tolerance, genoprettelsesegnethed, overensstemmelse		X
Brugervenlighed	Forståelsesegnethed, læringsegnethed, brugsegnethed, attraktivitet, overensstemmelse	X	
Effektivitet	Performance (tidsadfærd), resurseudnyttelse, overensstemmelse		X
Vedligeholdelses-egnethed	Analyserbarhed, ændringsegnethed, stabilitet, testbarhed, overensstemmelse		X
Flytbarhed	Tilpasningsegnethed, installerbarhed, sameksistens, udskiftningsegnethed, overensstemmelse		X

Testanalytikeren bør koncentrere sig om kvalitetskarakteristikker for funktionalitet og brugervenlighed. Adgangstest bør også foretages af testanalytikeren. Selv om adgangsegnethed ikke er anført som en under-karakteristik, betragtes det ofte som værende en del af brugervenlighedstest. Test af de andre kvalitetskarakteristikker betragtes sædvanligvis som værende en del af den teknisk testanalytikers ansvar. Mens denne arbejdsfordeling kan variere for forskellige organisationer, er det den, der følges i disse ISTQB pensa.

Under-karakteristikken overensstemmelse er vist for alle kvalitetskarakteristikkerne. I visse sikkerhedskritiske eller regulerede miljøer kan hver kvalitetskarakteristik også skulle stemme overens med specifikke standarder og regler/love (f.eks. kan funktionalitetsoverensstemmelse vise at funktionaliteten stemmer overens med en specifik standard, som f.eks. brug af en særlig kommunikationsprotokol for at kunne sende og modtage data fra en chip). Da disse standarder kan variere voldsomt afhængig af industrien, vil de ikke blive behandlet i dybden her. Hvis testanalytikeren arbejder i et miljø, for hvilket der gælder overensstemmelseskrav, er det vigtigt at forstå disse krav og at sikre, at både testen og testdokumentationen vil opfylde overensstemmelseskravene.

Man skal identificere de typiske risici for alle kvalitetskarakteristikker og under-karakteristikker der behandles i dette afsnit for at man kan formulere og dokumentere en passende teststrategi. Test af kvalitetskarakteristikker kræver særlig opmærksomhed på livscyklus-timing, nødvendige værktøjer og adgang til software, dokumentation og teknisk ekspertise. Uden planlægning af en strategi til at håndtere hver karakteristik og dens unikke testbehov har testerne måske ikke tilstrækkelig tid i tidsplanen til planlægning, klargøring og testafvikling. Noget af denne test, f.eks. brugervenlighedstest, kan kræve allokering af specielle menneskelige ressourcer, ekstensiv planlægning, dedikerede laboratorier, specifikke værktøjer, specialiserede testevner og, i de fleste tilfælde, en betydelig mængde tid. I nogle tilfælde kan det være en særlig gruppe af brugervenligheds- eller brugererfaringseksperter der gennemfører brugervenlighedstest.

Test af kvalitetskarakteristikker og under-karakteristikker skal være integreret i den overordnede testtidsplan med tildeling af tilstrækkelige ressourcer til opgaven. Hvert af disse områder har specifikke behov, angriber specifikke problemer og kan forekomme på forskellige tidspunkter i softwareudviklingslivscyklussen, som beskrevet i de følgende afsnit.

Selv om testanalytikeren måske ikke er ansvarlig for de kvalitetskarakteristikker, der kræver en mere teknisk tilgang, er det vigtigt, at testanalytikeren er opmærksom på de andre karakteristikker og forstår de overlappende testområder. For eksempel vil et produkt, der fejler i performancetest, sandsynligvis også fejle i brugervenlighedstest, hvis det er for langsomt til, at brugeren kan bruge det effektivt. På samme måde vil et produkt med tværoperationalitetsproblemer for nogle komponenter sandsynligvis ikke være klar til flytbarhedstest, da dette vil have tendens til at skjule de mere basale problemer, når miljøet ændres.

4.2 Test af forretningsdomæner

Funktionel test er et vigtigt fokus for testanalytikeren. Funktionel test er fokuseret på "hvad" produktet gør. Testgrundlaget for funktionel test er generelt et kravs- eller specifikationsdokument, specifik domæneekspertise eller implicitte behov. Funktionelle tests varierer i forhold til det testniveau, som de udføres i, og kan også være påvirket af softwareudviklingslivscyklussen. For eksempel vil en funktionel test udført under integrationstest teste funktionaliteten af forbundne moduler, som implementerer en enkelt defineret funktion. På systemtestniveauet omfatter funktionel test test af applikationens funktionalitet som et hele. For systemer-af-systemer vil funktionel test fortrinsvis fokusere på end-to-end test hen over de integrerede systemer. I et agilt miljø er funktionel test normalt begrænset til den funktionalitet, der er stillet til rådighed i en specifik iteration eller sprint, selvom regressionstest for en iteration kan dække al frigivet funktionalitet.

Der anvendes et stort antal forskellige testteknikker under funktionel test (se Kapitel 3). Funktionel test kan udføres af en dedikeret tester, en domæneekspert eller en udvikler (normalt på komponentniveauet).

I tillæg til den funktionelle test dækket i dette afsnit, er der også to kvalitetskarakteristikker, der er en del af testanalytikerens ansvarsområde, der anses for at være ikke-funktionelle (fokuseret på "hvordan" produktet leverer funktionaliteten) testområder. Disse to ikke-funktionelle attributter er brugervenlighed og adgang.

Følgende kvalitetskarakteristikker behandles i dette afsnit:

Funktionelle kvalitetsunder-karakteristikker
- Nøjagtighed
- Egnethed
- Tværoperationalitet.

Ikke-funktionelle kvalitetskarakteristikker

- Brugervenlighed
- Adgang.

4.2.1 Nøjagtighedstest

Funktionel nøjagtighed involverer test af applikationens overholdelse af specificerede eller implicitte krav og kan også omfatter beregningsnøjagtighed. Nøjagtighedstest anvender mange af de testteknikker, der er beskrevet i kapitel 3, og bruger ofte specifikationen eller et ældre system som testorakel. Nøjagtighedstest kan foretages på ethvert trin i livscyklussen og er målrettet mod ukorrekt behandling af data eller situationer.

4.2.2 Egnethedstest

Egnethedstest involverer evaluering og validering af hensigtsmæssigheden af et sæt af funktioner ift. dets tilsigtede opgaver. Denne test kan være baseret på usecases. Egnethedstest bliver normalt udført under systemtesten, men kan også udføres under de senere integrationstestfaser. Defekter opdaget i denne test er indikationer af at systemet ikke vil kunne opfylde brugerens behov på en acceptabel måde.

4.2.3 Tværoperationalitetstest

Tværoperationalitetstest tester i hvilken grad to eller flere systemer eller komponenter kan udveksle information og efterfølgende anvende den information, der er blevet udvekslet. Test skal dække alle de påtænkte målmiljøer (inkl. variationer i hardwaren, software, middleware, operativsystem etc.) for at sikre, at dataudvekslingen vil fungere ordentligt. I realiteten er dette kun opnåeligt for et relativt lille antal miljøer. Derfor kan tværoperationalitetstest være begrænset til en repræsentativ gruppe af miljøer. Specifikation af tests for tværoperationalitet kræver, at kombinationerne af de påtænkte målmiljøer er identificerede, konfigurerede og til rådighed for testteamet. Disse miljøer testes så ved brug af et udvalg af funktionelle testcases, som berører de forskellige dataudvekslingspunkter, der er til stede i miljøet.

Tværoperationalitet relaterer til, hvordan forskellige softwaresystemer interagerer med hinanden. Software med gode tværoperationalitetskarakteristikker kan integreres med et antal andre systemer, uden at det kræver omfattende ændringer. Antallet af ændringer og den indsats, der kræves for at udføre disse ændringer, kan bruges som et mål for tværoperationalitet.

Test af software-tværoperationalitet kan f.eks. fokusere på følgende designfeatures:

- Brug af generelle industrikommunikationsstandarder, som f.eks. XML
- Evne til automatisk at bestemme kommunikationsbehovene for systemer, der interageres med, og tilpasse sig hertil.

Tværoperationalitetstest kan være af speciel betydning for organisationer, der udvikler ”Commercial Off The Shelf” (COTS) software og værktøjer, og organisationer, der udvikler systemer-af-systemer.

Denne type af test gennemføres under komponentintegrations- og systemtest med fokus på systemets interaktion med sit miljø. På systemintegrationsniveauet udføres denne type test for at bestemme, hvor godt det fuldt udviklede system interagerer med andre systemer. Da systemer kan interagere på flere niveauer, må testanalytikeren forstå disse interaktioner og kunne skabe de betingelser, hvorunder de forskellige interaktioner vil blive anvendt. For eksempel, hvis to systemer vil udveksle data, må testanalytikeren kunne skabe de nødvendige data og de transaktioner, der skal til for at gennemføre dataudvekslingen. Det er vigtigt at huske, at det måske ikke er alle interaktioner,

der er specificeret i kravdokumenterne. I stedet vil mange af disse interaktioner kun blive defineret i systemarkitektur- og designdokumenterne. Testanalytikeren må være i stand til og villig til at læse disse dokumenter for at bestemme, hvor der sker informationsudveksling mellem systemer og mellem systemet og dets miljø for at sikre, at alt bliver testet. Teknikker som f.eks. beslutningstabeller, tilstandsovergangsdiagrammer, usecases og kombinatorisk test kan alle bruges i tværoperationalitetstest. Typiske defekter, der findes, omfatter ukorrekt dataudveksling mellem interagerende komponenter.

4.2.4 Brugervenlighedstest

Det er vigtigt at forstå, hvorfor brugere kan have svært ved at bruge systemet. For at opnå denne forståelse er det først nødvendigt at forstå, at begrebet "bruger" kan dække over mange forskellige typer af personer, strækkende sig fra IT-eksperter til børn og personer med handicap.

Nogle nationale institutioner (f.eks. British Royal National Institute for the Blind) anbefaler, at websider skal være tilgængelige for handicappede, blinde, svagsynede, bevægelighedshæmmede, døve og erkendelseshæmmede brugere. Det kan også forbedre brugervenligheden for andre hvis man kontrollerer at applikationer og websider er anvendelige for sådanne brugere. Tilgængelighed beskrives yderligere nedenfor.

Brugervenlighedstest tester, hvor let brugere kan bruge eller lære at bruge systemet til at nå et specifikt mål i en specifik sammenhæng. Brugervenlighedstest er rettet mod måling af:

- Egnethed – softwareproduktets evne til at brugerne kan opnå bestemte mål med nøjagtighed og fuldstændighed i en specificeret brugssammenhæng
- Effektivitet – produktets evne til at gøre det muligt for brugere at bruge passende mængder af ressourcer i forhold til den effektivitet, der er opnået i en specificeret brugssammenhæng
- Tilfredshed – softwareproduktets evne til at tilfredsstille brugerne i en specificeret brugssammenhæng.

Attributter, der kan måles:

- Forståelsesegnethed - softwareattributter, der påvirker den indsats, det kræver af brugeren at forstå det logiske koncept og dets anvendelighed
- Læringsegnethed – softwareattributter, der påvirker den indsats, det kræver af brugeren at lære applikationen at kende
- Brugsegnethed - softwareattributter, der påvirker den indsats, det kræver af brugeren at gennemføre opgaver effektivt og godt
- Attraktivitet – softwarens evne til at gøre brugeren glad.

Brugervenlighedstest bliver normalt udført i to trin:

- Informativ brugervenlighedstest – test der udføres iterativt i design- og prototypefaserne for at støtte (eller "forme") designet ved at finde defekter i brugervenlighedsdesignet
- Opsummerende brugervenlighedstest - test der udføres efter implementering for at måle brugervenligheden og finde problemer i en færdig komponent eller et færdigt system.

Brugervenlighedstesteres færdigheder bør omfatte ekspertise i eller kendskab til de følgende områder:

- Sociologi
- Psykologi
- Overholdelse af nationale standarder (inkl. tilgængelighedsstandarder)
- Ergonomi.

4.2.4.1 Gennemførelse af brugervenlighedstests

Validering af den aktuelle implementering bør foretages under betingelser, der er så tæt som muligt på dem, som systemet vil blive brugt under. Dette kan omfatte opsætning af et brugervenlighedslaboratorium med videokameraer, modelkontorer, review-paneler, brugere etc., så udviklere kan observere det aktuelle systems effekt på rigtige mennesker. Formelle brugervenlighedstest kræver ofte en vis forberedelse af "brugerne" (disse kan være rigtige brugere eller brugerrepræsentanter) enten ved at give dem faste beskrivelser eller instruktioner, som de kan følge. Andre frie former for test giver brugeren mulighed for at eksperimentere med softwaren, så observatørerne kan fastslå, hvor nemt eller besværligt det er for brugeren at finde ud af, hvordan de skal gennemføre deres opgaver.

Mange brugervenlighedstests kan afvikles af testanalytikeren som en del af andre tests, for eksempel under funktionel systemtest. For at opnå en konsistent tilgang til opdagelse og rapportering af brugervenlighedsdefekter i alle faser af livscyklussen, kan brugervenlighedsguidelines være gavnlige. Uden brugervenlighedsguidelines kan det være svært at bestemme, hvad der er "uacceptabel" brugervenlighed. For eksempel, er det urimeligt for en bruger at skulle foretage 10 museklik for at logge in på en applikation? Uden specifikke guidelines, kan testanalytikeren komme i den vanskelige situation af skulle forsvare fejlrapporter, som udvikleren ønsker at lukke, fordi software fungerer "som designet". Det er meget vigtigt at have verificerbare brugervenlighedsspecifikationer defineret i kravene, såvel som at have et sæt brugervenlighedsguidelines, der gælder for alle sammenlignelige projekter. Sådanne guidelines bør omfatter emner som tilgængelighed af instruktioner, klarhed af prompts, antal af klik for at gennemføre en aktivitet, fejlmelding, arbejdsindikatorer (en form for indikator til brugeren om at systemet arbejder og ikke kan modtage yderligere input på dette tidspunkt), skærmdesignguidelines, brug af farver og lyd og andre faktorer, der påvirker brugerens oplevelse.

4.2.4.2 Specifikation af brugervenlighedstest

Vigtige teknikker for brugervenlighedstest er:

- Inspektion, evaluering eller review
- Dynamisk interaktion med prototyper
- Verificering og validering af den faktiske implementering
- Anvendelse af undersøgelser og spørgeskemaer.

Inspektion, evaluering eller review

Inspektion eller review af kravspecifikation og design fra et brugervenlighedsperspektiv, der øger brugerens involveringsniveau, kan være omkostningseffektive ved at problemer findes tidligt. Heuristisk evaluering (systematisk inspektion af et brugergrænsefladedesign for brugervenlighed) kan anvendes til at finde brugervenlighedsproblemer i designet, så disse kan blive behandlet som en del af en iterativ designproces. Dette omfatter, at et lille udvalg af evaluatorer undersøger grænsefladen og bedømmer dens overensstemmelse med kendte brugervenlighedsprincipper ("heuristikker"). Reviews er mere effektive, når brugergrænsefladen er mere synlig. For eksempel er prøve-skærmbilleder normalt nemmere at forstå og tolke end en tekstuel beskrivelse af den funktionalitet, der skal være i et skærmbillede. Visualisering er vigtigt for et dækkende brugervenlighedsreview af dokumentationen.

Dynamisk interaktion med prototyper

Når prototyper udvikles, bør testanalytikeren arbejde med prototyperne og hjælpe udviklerne med at udvikle prototypen ved at indarbejde brugerfeedback i designet. På denne måde kan prototyper forbedres og brugeren kan få et mere realistisk billede af, hvordan det fædige produkt vil se ud og føles.

Verifikation og validering af den aktuelle implementering
Hvor kravene specificerer brugervenlighedskarakteristikker for softwaren (f.eks. antallet af museklik for at opnå et specifikt mål), må der udarbejdes testcases for at verificere, at softwaren har disse egenskaber.

For at gennemføre validering af den faktiske implementering kan tests specificeret til funktionel systemtest udvikles til scenarier for brugervenlighedstest. Disse testscenarier måler specifikke brugervenlighedskarakteristikker, som f.eks. læringsegnethed eller brugsegnethed, snarere end de funktionelle resultater.

Testscenarier for brugervenlighed kan anvendes til specifikt at teste syntaks og semantik. Syntaks er strukturen eller grammatikken for grænsefladen (f.eks. hvad der kan indtastes i et felt), hvorimod semantik beskriver meningen og formålet (f.eks. fornuftige og meningsfulde systemmeddelelser og output til brugeren) med grænsefladen.

Black-box teknikker (for eksempel dem, der er beskrevet i Afsnit 3.2), specielt usecases, som kan defineres i almindelig tekst eller med UML (Unified Modeling Language), bruges nogle gange i brugervenlighedstest.

Testscenarier for brugervenlighedstest skal også omfatte brugerinstruktioner, afsat tid til interviews før og efter testen til at give instruktioner og få feedback og en godkendt protokol for gennemførelse af sessionerne. Denne protokol omfatter en beskrivelse af, hvordan testen vil blive foretaget, tider, noter og logning af sessioner og de interview- og undersøgelsesmetoder, der skal anvendes.

Anvendelse af undersøgelser og spørgeskemaer
Undersøgelses- og spørgeskemateknikker kan anvendes til at indsamle observationer og feedback angående brugeradfærd for systemet. Standardiserede og offentligt tilgængelige undersøgelser som f.eks. SUMI (Software Usability Measurement Inventory) og WAMMI (Website Analyse og MeasureMent Inventory) giver mulighed for benchmarking i forhold til en database med tidligere brugervenlighedsmålinger. Ud over det kan SUMI, der leverer konkrete brugervenlighedsmålinger, stille et sæt af færdiggørelses-/ acceptkriterier til rådighed.

4.2.5 Adgangstest

Det er vigtigt at overveje adgang til software for personer med specifikke behov eller begrænsninger i forhold til dets brug. Dette omfatter personer med handicap. Adgangstest bør tage hensyn til relevante standarder, som f.eks. the Web Content Accessibility Guidelines, og lovgivning, så som Disability Discrimination Acts (UK, Australia) og Section 508 (US). Adgangsvenlighed, ligesom brugervenlighed, skal overvejes i designfaserne. Test forekommer ofte under integrationsniveauerne og forsætter gennem systemtest- og ind i accepttestniveauerne. Defekter findes normalt, når softwaren ikke opfylder udpegede bestemmelser eller standarder, der er defineret for softwaren.

5 Reviews - 165 min.

Nøgleord
Ingen

Læringsmål for reviews

5.1 Introduktion

TA-5.1.1 (K2) Forklar hvorfor forberedelse af reviews er vigtigt for testanalytikere.

5.2 Brug af checklister i reviews

TA-5.2.1 (K4) Analyser en usecase eller en brugergrænseflade og identificer problemer i henhold til checklisteinformation givet i pensumbeskrivelsen.

TA-5.2.2 (K4) Analyser en kravspecifikation eller user story og identificer problemer i henhold til checklisteinformation givet i pensumbeskrivelsen.

5.1 Introduktion

En vellykket reviewproces kræver planlægning, deltagelse og opfølgning. Testanalytikere må være aktive deltagere i reviewprocessen og må bidrage med deres særlige synsvinkel. De bør have formel reviewtræning for bedre at forstå deres respektive roller i enhver reviewproces. Alle reviewdeltagere må være overbevist om fordelene ved et velgennemført review. Korrekt gennemførte reviews kan være det mest kost-effektive bidrag til den samlede, leverede kvalitet.

Uanset typen af review, der bliver gennemført, skal testanalytikeren afsætte tilstrækkelig tid til forberedelse. Dette omfatter tid til at reviewe arbejdsproduktet, tid til at kontrollere krydsrefererede dokumenter for at verificere konsistens og tid til at finde ud af, hvad der kan mangle i arbejdsproduktet. Uden tilstrækkelig forberedelsestid kan testanalytikeren være begrænset til kun at editere det, der allerede er i dokumentet, snarere end at deltage i et effektivt review, der maksimerer brugen af reviewteamets tid og giver den bedst mulige feedback. Et godt review omfatter forståelse af, hvad der er skrevet, bestemmelse af, hvad der mangler, og verifikation af, at det beskrevne produkt er konsistent med andre produkter, der enten allerede er udviklet eller er under udvikling. For eksempel, når testanalytikeren reviewer en integrationstestplan, skal han eller hun også overveje de objekter, der bliver integreret. Hvad er de nødvendige betingelser for at de er klar til integration? Er der afhængigheder, der skal dokumenteres? Er der data til rådighed til at teste integrationspunkterne? Et review er ikke isoleret til det arbejdsprodukt, der bliver reviewet, det skal også overveje interaktionen mellem objektet og andre objekter i systemet.

Det er nemt for forfatteren af et produkt, der bliver reviewet, at føle sig kritiseret. Testanalytikeren skal gøre sig umage med at behandle enhver reviewkommentar fra det synspunkt, at der arbejdes sammen med forfatteren om at skabe det bedst mulige produkt. Ved at benytte denne tilgang vil kommentarer bliver udformet konstruktivt og vil være rettet mod arbejdsproduktet og ikke mod forfatteren. For eksempel, hvis en sætning er tvetydig, er det bedre at sige “Jeg forstår ikke, hvad jeg skal teste for at verificere, at dette krav er blevet implementeret korrekt, Kan du hjælpe mig med at forstå det?” snarere end “Dette krav er tvetydigt og ingen vil kunne regne det ud.” Testanalytikerens opgave i et review er at sikre, at den information, der er indeholdt i et arbejdsprodukt, vil være tilstrækkelig til at støtte testarbejdet. Hvis informationen ikke er til stede, ikke er klar, eller ikke er på et tilstrækkeligt detaljeringsniveau, så er dette sandsynligvis en defekt, der skal rettes af forfatteren. Ved at opretholde en positiv tilgang snarere end en kritisk tilgang, vil kommentarer blive bedre modtaget, og mødet vil være mere produktivt.

5.2 Brug af checklister i reviews

Checklister bruges ved reviews til at minde deltagerne om at kontrollere specifikke punkter under reviewet. Checklister kan også medvirke til at afpersonalisere reviewet, f. eks., "Dette er den samme checkliste, som vi bruger ved alle reviews, vi går ikke specielt efter jeres arbejdsprodukt." Checklister kan være generiske og anvendt til alle reviews, eller de kan fokusere på specifikke kvalitetskarakteristikker, områder eller typer af dokumenter. For eksempel, kan en generisk checkliste verificere de generelle dokumentegenskaber, som f.eks. eksistens af en entydig identifikation, ingen TBD referencer, korrekt formatering og tilsvarende overensstemmelsesområder. En specifik liste for et kravdokument kan indeholde kontrol af korrekt brug af termerne “skal” og “kan”, kontrol af testbarhed af hvert krav osv. Kravenes format kan også indikere, hvilken type checkliste, der kan bruges. Der vil gælde andre reviewkriterier for et kravdokument, der er i fortællende tekstformat, end for et, der er diagrambaseret.

Checklister kan også være orienteret mod programmør- arkitekt- eller testkompetencer. I testanalytikerens tilfælde vil en checkliste for testkompetencer være det mest passende. Sådanne checklister kan omfatter emner som vist nedenfor.

Checklister anvendt til krav, usecases og user stories har generelt et andet fokus end dem, der bruges til kode eller arkitektur. Kravorienterede checklister kan omfatte følgende emner:

- Testbarhed af de enkelte krav
- Acceptkriterier for de enkelte krav
- Struktur for usecases, hvis det er relevant
- Entydig identifikation af hvert krav/usecase/user story
- Versioning af hvert krav/usecase/user story
- Sporbarhed til hvert krav fra forretnings- og marketingskrav
- Sporbarhed mellem krav og usecases.

Listen oven for er kun ment som et eksempel. Det er vigtigt at huske, at hvis et krav ikke er testbart, dvs. at det er defineret på en sådan måde, at testanalytikeren ikke kan bestemme, hvordan det kan testes, så er der en defekt i kravet. For eksempel, er et krav, der lyder "Softwaren børe være meget brugervenlig" ikke testbart. Hvordan kan testanalytikeren fastslå, om softwaren er brugervenlig eller endda meget brugervenlig? Hvis kravet i stedet lød "Softwaren skal overholde de brugervenlighedsstandarder, der er beskrevet i brugervenlighedsstandarddokumentet", og hvis brugervenlighedsstandarddokumentet virkelig eksisterede, så er dette et testbart krav. Det er også et overordnet krav, fordi dette ene krav gælder for alle elementer i grænsefladen. I dette tilfælde kunne dette ene krav nemt give anledning til mange testcases i en indviklet applikation. Sporbarhed fra dette krav, eller måske fra brugervenlighedsstandarddokumentet til testcases er også kritisk, for hvis den refererede brugervenlighedsspecifikation skulle ændre sig, vil alle testcases skulle reviewes og om nødvendigt opdateres.

Et krav er heller ikke testbart, hvis testeren ikke er i stand til at bestemme, om testen er bestået eller fejlet, eller ikke er i stand til at konstruere en test, der kan bestå eller fejle. For eksempel, "Systemet skal være til rådighed 100 % af tiden, 24 timer pr. dag, 7 dage om ugen, 365 (eller 366) dage om året" er ikke testbart.

.

En simpel checkliste for reviews af usecases kan omfatte:

- Er hovedforløbet (scenariet) klart defineret?
- Er alle alternative forløb (scenarier) identificeret, inkl. fejlhåndtering?
- Er alle brugergrænseflade-meddelelser defineret?
- Er der kun ét hovedforløb? (Sådan bør det være - ellers er der flere usecases!)
- Er alle forløb testbare?

En simpel checkliste for brugervenlighed af brugergrænsefladen for en applikation kan omfatte:

- Er hvert felt og dets funktion defineret?
- Er alle fejlmeddelelser defineret?
- Er alle brugerprompts defineret og konsistente?
- Er felternes tabulatorrækkefølge defineret?
- Er der tastatur-alternativer til musehandlinger?
- Er genvejstastekombinationer defineret for brugeren (f.eks. klip og sæt ind)?
- Er der afhængigheder mellem felter (som f.eks. det, at en særlig dato skal være senere end en anden dato)?
- Er der et skærmlayout?
- Passer skærmlayoutet til de specificerede krav?
- Er der en brugerindikator, der vises, når systemet arbejder?
- Opfylder skærmen minimumklik-kravet (hvis det er defineret)?
- Vil navigationen opleves logisk for en bruger?
- Opfylder skærmbillederne eventuelle krav til læringsegnethed?

- Er der hjælpetekst til rådighed for brugeren?
- Er der hjælpetekst til rådighed for brugeren, når musen holdes henover (tooltips)?
- Vil brugeren opfatte dette som "attraktivt" (subjektiv vurdering)?
- Er farvebrug konsistent med andre applikationer og med organisationens standarder?
- Er lydeffekterne passende, og er de konfigurerbare?
- Opfylder skærmen lokaliseringskrav?
- Kan brugeren finde ud af, hvad der skal gøres (forståelsesegnethed) (subjektiv vurdering)?
- Vil brugeren kunne huske, hvad der skal gøres (læringsegnethed) (subjektiv vurdering)?

I et agilt projekt er kravene normal udtrykt i user stories. Disse historier repræsenterer en lille smule demonstrerbar funktionalitet. Mens en usecase er en brugertransaktion, der gennemløber flere funktionalitetsområder, er en user story mere isoleret og er generelt afgrænset af den tid, det tager at udvikle den. En checkliste for en user story kan omfatte:

- Er historien passende i forhold til målet for iterationen/sprintet?
- Er acceptkriterierne defineret og testbare?
- Er funktionaliteten klart defineret?
- Er der nogen afhængigheder mellem denne user story og andre?
- Er denne user story prioriteret?
- Indeholder historien én funktionalitet?

Hvis storyen definerer en ny grænseflade, vil brug af en generisk story-checkliste (som den oven for) og en detaljeret brugergrænseflade-checkliste være relevant.

En checkliste kan tilrettes baseret på:

- Organisation (f.eks. vedrørende firmapolitikker, standarder, konventioner)
- Projekt- eller udviklingsindsats (f.eks. fokus, tekniske standarder, risici)
- Reviewobjektet (f.eks. kan kodereview tilrettes til specifikke programmeringssprog).

Gode checklister vil finde problemer og vil også hjælpe til at starte diskussioner vedr. andre objekter, der ikke er blevet specifikt refereret i listen. En kombination af forskellige checklister kan sikre, at et review giver arbejdsprodukter af højeste kvalitet. Både standard-checklister som nævnt i pensum for Foundation-niveauet og organisationsspecifikke checklister som vist ovenfor vil hjælpe testanalytikeren til effektive reviews.

For mere information om review og inspektion se [Gilb93] og [Wiegers03].

6 Fejlhåndtering – 120 min.

Nøgleord
Defekttaksonomi, faseinddæmning, analyse af underliggende årsag

Læringsmål for fejlhåndtering

6.2 Hvornår kan man finde en defekt?
TA-6.2.1 (K2) Forklar hvordan faseinddæmning kan mindske omkostningerne.

6.3 Oplysninger i fejlrapporten
TA-6.3.1 (K2) Forklar den information, der kan være behov for, når en ikke-funktionel defekt dokumenteres.

6.4 Klassifikation af defekter
TA-6.4.1 (K4) Identificér, indsaml og registrér klassifikationsinformation for en given defekt.

6.5 Analyse af underliggende årsager
TA-6.5.1 (K2) Forklar formålet med analyse af underliggende årsager.

6.1 Introduktion

Testanalytikere vurderer systemets adfærd ud fra forretnings- og brugerbehov, f. eks. ved brugeren, hvad han eller hun skal gøre i forhold til en bestemt besked eller systemadfærd. Ved at sammenligne det faktiske med det forventede resultat, finder testanalytikeren ud af, om systemet opfører sig korrekt. En uregelmæssighed (også kaldet en afvigelse) er en uventet hændelse, der kræver nærmere undersøgelse. En uregelmæssighed kan være en afvigelse forårsaget af en defekt. En uregelmæssighed kan eller kan ikke resultere i oprettelsen af en fejlrapport. En defekt er et faktisk problem, der bør løses.

6.2 Hvornår kan man finde en defekt?

En defekt kan findes ved statisk test og symptomerne af defekten, afvigelsen, kan findes ved dynamisk test. Hver fase i softwarens udviklingslivscyklus bør have metoder til at finde og eliminere potentielle afvigelser. For eksempel, bør reviews af kode og design bruges i udviklingsfasen for at finde defekter. Under dynamisk test, bruges testcases til at finde afvigelser.

Jo tidligere en defekt findes og udbedres, jo lavere er prisen for kvalitet for systemet som helhed. For eksempel kan statisk test finde defekter før dynamisk test er mulig. Dette er en af grundene til, at statisk test er en kost-effektiv tilgang til produktion af højkvalitetssoftware.

Fejlopfølgningsværktøjet bør give testanalytikeren mulighed for at registrere den fase i livscyklussen, hvor defekten blev introduceret, og den fase, som den blev fundet i. Hvis de to faser er den samme, er der opnået perfekt faseinddæmning. Dette betyder, at defekten blev introduceret og fundet i den samme fase, og ikke ”undslap” til en senere fase. Et eksempel på dette ville være, at man identificerer et ukorrekt krav under kravreviewet og retter det med det samme. Ikke alene er dette en effektiv brug af kravreviewet, det sparer også store udgifter for organisationen. Hvis et ukorrekt krav “undslipper” i kravreviewet og senere implementeres af udvikleren, testes af testanalytikeren, og fanges af en bruger under brugeraccepttest, har alt arbejde udført for at opfylde kravet været spild af tid - foruden at brugeren mister tillid til systemet.

Faseinddæmning er en effektiv måde at nedsætte omkostningerne ved defekter på.

6.3 Oplysninger i fejlrapporten

Formålet med de felter (parametre), der er til rådighed i en fejlrapport, er at give nok information til, at fejlrapporten er handlingsrettet. En handlingsrettet fejlrapport er:

- Komplet – alle de nødvendige informationer er i rapporten
- Koncis – der er ingen irrelevante informationer i rapporten
- Akkurat - informationerne i rapporten er korrekte og beskriver klart det forventede og det faktiske resultat, så vel som de trin, der skal til for at genskabe afvigelsen
- Objektiv – rapporten er en professionelt skrevet liste af fakta.

Den information, der er registreret i en fejlrapport bør inddeles i datafelter. Jo mere veldefinerede felterne er, jo nemmere er det at rapportere de enkelte defekter, så vel som at producere tendensrapporter og andre overbliksrapporter. Når et defineret antal muligheder er til rådighed for et felt med dropdownlister med de tilgængelige værdier, kan det nedsætte den tid, der er nødvendig for at registrere en defekt. Dropdownlister er kun effektive, når antallet af muligheder er begrænset, og brugeren ikke behøver at scrolle gennem en lang liste for at finde den korrekte mulighed. Forskellige typer af fejlrapporter kræver forskellige informationer og fejlhåndteringsværktøjet bør være fleksibelt nok til at prompte for de korrekte felter afhængig af defekttypen. Data bør registreres i forskellige

felter, ideelt set understøttet af datavalidering for at undgå indtastningsfejl og sikre effektiv rapportering.

Fejlrapporter skrives for afvigelser opdaget under funktionel og ikke-funktionel test. Informationer i en fejlrapport bør altid være rettet mod klart at identificere det scenarie, hvori problemet blev opdaget, inklusiv trin og data, der er nødvendige for at reproducere scenariet, så vel som de forventede og faktiske resultater. Ikke-funktionelle fejlrapporter kan kræve flere detaljer mht. miljøet, andre performance-parametre (f.eks. belastningens størrelse), rækkefølge af trin og forventede resultater. Når en brugervenlighedsafvigelse dokumenteres, er det vigtigt at skrive, hvad brugeren forventede, at softwaren ville gøre. For eksempel, hvis brugervenlighedsstandarden er, at en operation skulle gennemføres med mindre end fire museklik, bør fejlrapporten oplyse, hvor mange klik, der var nødvendige i forhold til den nævnte standard. I de tilfælde hvor en standard ikke er tilgængelig, og kravene ikke dækkede softwarens ikke-funktionelle kvalitetsaspekter, kan testerne bruge den "fornuftige person"-test til at bestemme, at brugervenligheden er uacceptabel. I det tilfælde skal den "fornuftige persons" forventninger klart fremgå af fejlrapporten. Da ikke-funktionelle krav nogle gange mangler i kravdokumentationen, giver dokumentation af ikke-funktionelle afvigelser flere udfordringer for testerne end dokumentering af den "forventede" i forhold til den "faktiske" adfærd.

Mens det almindelige formål med at skrive en fejlrapport er at få en løsning på problemet, skal defektinformationer også gives for at støtte præcis klassifikation, risikoanalyse og procesforbedring.

6.4 Klassifikation af defekter

Der er flere klassifikationsniveauer, som en fejlrapport kan få i løbet af sin livscyklus. Korrekt defektklassifikation er en integreret del af korrekt fejlrapportering. Klassifikationer bruges til at gruppere defekter, at vurdere testens effektivitet, at vurdere udviklingslivscyklussen effektivitet og at finde interessante trends.

Almindelig klassifikationsinformation for ny-identificerede defekter:

- Projektaktivitet der resulterede i at defekten blev opdaget – f.eks. review, audit, inspektion, programmering, test
- Projektfase som defekten blev introduceret i (hvis kendt) – f.eks. krav, design, detaljeret design, programmering
- Projektfasen som defekten blev opdaget i – f.eks. krav, design, detaljeret design, programmering, kode-review, modultest, integrationstest, systemtest, accepttest
- Formodet årsag til defekten – f.eks. krav, design, grænseflade, kode, data
- Gentagelighed – f.eks. en gang, nogle gange, gentagelig
- Symptom – f.eks. nedbrud, hængende, brugergrænsefladefejl, systemfejl, performance.

Når defekten er blevet undersøgt, kan yderligere klassifikation være mulig:

- Den underliggende årsag - den fejltagelse, der blev begået, og som resulterede i defekten, f.eks. proces, programmeringsfejl, brugerfejl, testfejl, konfigureringsproblem, dataproblem, tredje-parts software, eksternt softwareproblem, dokumentationsproblem
- Kilde – det arbejdsprodukt som fejltagelsen blev begået i, f.eks., krav, design, detaljeret design, arkitektur, databasedesign, brugerdokumentation, testdokumentation
- Type – f.eks. logisk problem, beregningsproblem, tidsproblem, datahåndtering, forbedring.

Når defekten er rettet (eller er blevet udskudt eller ikke er blevet godkendt), kan der være endnu mere klassifikationsinformation til rådighed, som f.eks.:

- Løsning – f.eks. kodeændring, dokumentationsændring, udskudt, ikke et problem, duplikat

- Korrigerende handling – f.eks. kravreview, kodereview, modultest, konfigurationsdokumentation, dataforberedelse, ikke ændring foretaget.

Ud over disse klassifikationskategorier, klassificeres defekter også ofte baseret på alvorsgrad og prioritet. Herudover, afhængig af projektet, kan det give mening at klassificere baseret på effekten på opgavens sikkerhed, effekten på projektets tidsplan, projektomkostninger, projektrisici og projektkvaliteten. Disse klassifikationer kan tages med i overvejelserne i beslutninger om, hvor hurtigt en løsning vil blive leveret.

Det sidste klassifikationsområde er den endelige løsning. Defekter er ofte samlet i grupper baseret på deres løsning, f.eks. rettet/verificeret, lukket/ikke et problem, udskudt, åben/uløst. Denne klassifikation bruges normalt gennem hele projektet, idet defekterne følges gennem deres livscyklus.

De klassifikationsværdier, en organisation bruger, er ofte tilpassede. Værdierne ovenfor er kun eksempler på nogle af de værdier, der almindeligvis bruges i industrien. Det er vigtigt at klassifikationsværdierne bruges konsistent, så de kan være anvendelige. For mange klassifikationsfelter vil gøre oprettelse og behandling af en defekt noget tidskrævende, så det er vigtigt at vægte værdien af at data bliver indsamlet i forhold til den stigende omkostning for hver defekt, der bliver behandlet. Muligheden for at tilpasse de klassifikationsværdier, der indsamles af et værktøj, er ofte en vigtigt faktor i valg af værktøj.

6.5 Analyse af underliggende årsager

Formålet med analyse af den underliggende årsag er at fastlægge, hvad der var årsag til, at defekten opstod, og at fremskaffe data til at støtte procesændringer, der vil fjerne underliggende årsager, der er ansvarlige for en betydelig del af defekterne. Analyse af underliggende årsag bliver normalt udført af den person, der undersøger og enten retter problemet eller bestemmer, om problemet ikke skal eller ikke kan løses. Dette er normalt udvikleren. Angivelse af en foreløbig underliggende årsag foretages normalt af testanalytikeren, som vil foretage et kvalificeret gæt mht., hvad der sandsynligvis har skabt problemet. Når rettelsen skal bekræftes, vil testanalytikeren verificere den underliggende årsag angivet af udvikleren. På det tidspunkt, hvor den underliggende årsag er fastsat, er det også almindeligt at bestemme eller bekræfte den fase, hvori defekten blev introduceret.

Typiske underliggende årsager:

- Uklart krav
- Manglende krav
- Forkerte krav
- Ukorrekt implementering af design
- Ukorrekt implementering af grænseflade
- Logiske kodefejl
- Beregningsfejl
- Hardware-fejl
- Grænseflade-fejl
- Invalide data.

De underliggende årsager bliver analyseret for at fastslå hvilke fælles problemer der har skabt defekter. Hvis mange defekter f.eks. skyldes uklare krav, så er der grund til at gøre mere ud af review af kravene. Hvis implementering af grænsefladen er et problem på tværs af flere udviklingsgrupper, så er der behov for fælles designmøder.

Information om underliggende årsager hjælper organisationen til at arbejde for effektive procesændringer og til at beregne de omkostninger ved defekter, der skyldes en specifik, underliggende årsag. Dette kan hjælpe med at skaffe penge til procesændringer, der kan kræve anskaffelse af flere værktøjer og udstyr så vel som ændringer i tidsplaner. Pensum for Expert Improving the Test Process [ISTQB_EL_ITP] behandler analyse af underliggende årsag i yderligere detaljer.

7 Testværktøjer - 45 min.

Nøgleord

Nøgleordsdrevet test, værktøj til testdataforberedelse, testdesignværktøj, testafviklingsværktøj

Læringsmål for testværktøjer

7.2 Testværktøjer og automatisering

TA-7.2.1 (K2) Forklar fordelene ved at bruge værktøjer til at forberede testdata, testdesignværktøjer og testafviklingsværktøjer.

TA-7.2.2 (K2) Forklar testanalytikerens rolle i nøgleordsdrevet automatisering.

TA-7.2.3 (K2) Forklar fremgangsmåden ved fejlfinding for en afvigelse fundet under afvikling af automatiseret test.

7.1 Introduktion

Et testværktøj kan forbedre effektivitet og præcision af testarbejdet væsentligt, hvis det er det rette værktøj, og det er implementeret rigtigt. Styring af organisationens testværktøjer er et aspekt, der hører med i en velfungerende testorganisation. Der er stor forskel på testværktøjernes anvendelighed, og markedet er under konstant forandring. Man kan anskaffe testværktøjer ved kommercielle leverandører eller fra hjemmesider med "freeware" eller "shareware" værktøj.

7.2 Testværktøjer og automatisering

Meget af en testanalytikers arbejde kræver effektiv brug af værktøjer. Viden om hvilke værktøjer, der skal bruges, og hvornår, kan øge testanalytikerens effektivitet og kan hjælpe med at give en bedre testdækning inden for den givne tid.

7.2.1 Testdesignværktøjer

Testdesignværktøjer bruges til at hjælpe med at oprette testcases og testdata, der skal bruges ved test. Disse værktøjer kan arbejde ud fra specifikke kravdokumentformater, modeller (f.eks. UML) eller inputs givet af testanalytikeren. Testdesignværktøjer er ofte designede og bygget til at arbejde med specifikke formater og specifikke produkter som f.eks. specifikke kravstyringsværktøjer.

Testdesignværktøjer kan give information, som testanalytikeren kan bruge til at fastlægge testtyper, der er nødvendige for at opnå det ønskede niveau af testdækning, tillid til systemet, eller reduktion af produktrisikoen. For eksempel vil et klassifikationstræ-værktøj kunne danne det sæt af kombinationer, der er nødvendigt for at opnå fuld dækning for et bestemt dækningskriterium. Med denne information kan testanalytikeren derefter fastlægge hvilke testcases, der skal afvikles.

7.2.2 Værktøjer til forberedelse af testdata

Værktøjer til forberedelse af testdata giver flere fordele. Nogle testdataforberedelsesværktøjer er i stand til at analysere et dokument, som f.eks. et kravdokument eller endda kildekode for at bestemme de data, der er nødvendige til testen for at opnå en dækningsgrad. Andre testdataforberedelses-værktøjer kan tage et datasæt fra et produktionssystem og “skrubbe” eller “anonymisere” det for at fjerne personlig information, mens den interne integritet i data bevares. De skrubbede data kan herefter bruges til test uden risiko for en sikkerhedslæk eller misbrug af personlig information. Dette er specielt vigtigt, når der kræves store mængder at data. Andre datagenererende værktøjer kan bruges til at generere testdata fra et givet sæt af inputparametre (dvs. til brug for vilkårlig test). Nogle af disse vil analysere databasestrukturen for at afgøre, hvilke inputs der er nødvendige fra testanalytikeren.

7.2.3 Automatiserede testafviklingsværktøjer

Testafviklingsværktøjer anvendes på alle testniveauer, og de anvendes mest af testanalytikere til at afvikle tests og kontrollere testens resultat. Formålet med at bruge testafviklingsværktøjer er typisk en eller flere af de følgende:

- at reducere omkostninger (i form af indsats og/eller tid)
- at afvikle flere tests
- at afvikle samme test i mange miljøer
- at gøre testafvikling mere gentagelig
- at afvikle tests, som det ville være umuligt at afvikle manuelt (dvs. store datavalideringstests).

Disse formål lapper ofte ind over hovedformålet, nemlig at øge dækning og samtidig reducere omkostninger.

7.2.3.1 Anvendelighed

Investeringsafkastet for testafviklingsværktøjer er normalt højest, når det er regressiontest, der automatiseres, på grund af det forventede lave vedligeholdelsesniveau og den gentagne afvikling af testene. Automatisering af smoke-tests kan også være en effektiv brug af automatisering på grund af den hyppige brug af testene, behovet for et hurtigt resultat, og, skønt vedligeholdelsesomkostningerne kan være højere, muligheden for at have en automatiseret måde at evaluere en build i et kontinuert integrationsmiljø.

Testafviklingsværktøjer bruges normalt på system- og integrationstestniveauerne. Nogle værktøjer, specielt API testværktøjer, kan også bruges på komponenttestniveauet. Udnyttelse af værktøjerne hvor de er mest anvendelige, vil hjælpe med til at forbedre investeringsafkastet.

7.2.3.2 Grundregler for testautomatiseringsværktøjer

Testafviklingsværktøjer virker ved at afvikle et sæt af instruktioner skrevet i et programmeringssprog, ofte kaldet er scriptsprog. Instruktionerne til værktøjet er på et meget detaljeret niveau, der specificerer input, rækkefølgen af inputtet og specifikke værdier for input og det forventede output. Dette kan gøre de detaljerede scripts følsomme overfor ændringer i softwaren under test (SUT), specielt når værktøjet interagerer med den grafiske brugergrænseflade (GUI).

De fleste testafviklingsværktøjer indeholder en sammenligner, der giver mulighed for at sammenligne et faktisk resultat med et gemt forventet resultat.

7.2.3.3 Implementering af testautomatisering

Tendensen i automatisering af testafvikling (som i programmering) er at bevæge sig fra detaljerede lavniveau-instruktioner til mere højniveausprog, brug af biblioteker, makroer og under-programmer. Designteknikker som f.eks. nøgleordsdreven og aktionsordsdreven opsamling af serier af instruktioner og referering af disse med et specielt "nøgleord" eller "aktionsord". Dette giver testanalytikeren mulighed for at skrive testcases i et menneskeligt sprog, idet man kan ignorere det underliggende programmeringssprog og lav-niveau funktioner. Brug af denne modulære skriveteknik giver nemmere vedligeholdelse ved ændringer i funktionaliteten og grænseflader for softwaren under test. [Bath08] Brug af nøgleord i automatiserede scripts behandles yderligere nedenfor.

Modeller kan bruges til at guide oprettelsen af nøgleordene eller aktionsordene. Ved at se på forretningsprocesmodeller i kravdokumenterne kan testanalytikeren fastlægge de centrale forretningsprocesser, der skal testes. Trinene i disse processer kan så bestemmes, inkl. de beslutningspunkter, der kan opstå i processerne. Beslutningspunkterne kan blive til aktionsord, som testautomatiseringen kan få og bruge fra nøgleords- eller aktionordsregneark. Modellering af forretningsprocesser er en metode til at dokumentere forretningsprocesserne og kan bruges til at identificere disse nøgleprocesser og beslutningspunkter. Modelleringen kan ske manuelt eller ved brug af værktøjer, der vil reagere ud fra input baseret på forretningsregler og procesbeskrivelser.

7.2.3.4 Forbedring af succesen med automatiseringsarbejdet

Når man bestemmer hvilke tests, der skal automatiseres, skal hver mulig testcase eller testsæt vurderes for at se, om den fortjener automatisering. Mange mislykkede automatiseringsprojekter er baseret på automatisering af allerede tilgængelige manuelle testcases uden at kontrollere den faktiske fordel ved automatisering. Det kan være optimalt for et givet sæt af testcases (et sæt) at indeholde manuelle, halv-automatiserede og fuldt automatiserede tests.

Følgende aspekter bør overvejes, når der implementeres et testautomatiseringsafviklingsprojekt:

Mulige fordele:

- Automatiseret testafviklingstid vil blive mere forudsigelig
- Regressionstest og defektvalidring ved brug af automatiserede tests vil være hurtigere og mere pålidelige sent i projektet

- Testernes eller testteamets status og tekniske viden kan blive forbedret ved brug af automatiserede værktøjer
- Automatisering kan være specielt brugbar ved iterative og inkrementelle udviklingslivscyklusser ved at give bedre regressionstest for hvert "build" eller iteration
- Dækning af særlige testtyper kan kun være mulig med automatiserede værktøjer (f.eks. store datavalideringsopgaver)
- Automatiseret testafvikling kan være mere omkostningseffektiv end manuel test for store mængder af datainput, konverterings- og sammenligningstest ved at give hurtig og konsistent input og verifikation.

Mulige risici:

- Ukomplette, ineffektive eller ukorrekte manuelle test kan blive automatiseret “som de er”
- Det kan være vanskeligt at vedligeholde testmaterialet, da det kræver mange ændringer, når softwaren under test ændres
- Testerne vil blive mindre involveret i testafviklingen, og dermed bliver der måske fundet færre defekter
- Testteamet kan have utilstrækkelige evner til at bruge de automatiserede værktøjer effektivt
- Irrelevante tests, der ikke bidrager til den overordnede testdækning kan blive automatiseret fordi de eksisterer og er stabile
- Tests kan blive uproduktive efterhånden som softwaren stabiliseres (pesticid paradokset).

Under brug af et testafviklingsværktøj, er det ikke altid klogt at automatisere manuelle testcases, som de er, men at redefinere testcasene for bedre brug af automatisering. Dette omfatter formatering af testcases, overvejelser om genbrugsmønstre, udvidelse af input ved brug af variable i stedet for brug af hårdtkodede værdier og brug af de fulde fordele af testværktøjer. Testafviklingsværktøjer har normalt mulighed for at gennemløbe mange tests, gruppere tests, gentage tests og ændre afviklingsrækkefølge, mens de også giver analyse- og rapporteringsfaciliteter.

For mange testafviklingsværktøjer er det nødvendigt med programmeringsevner for at skabe effektive tests (scripts) og testsuiter. Det er almindeligt, at store automatiserede testsuiter er meget svære at opdatere og styre, hvis det ikke er designet med omhu. Passende træning i testværktøjer, programmering og designteknikker er værdifuldt for at alle fordele ved værktøjerne bliver udnyttet.

Under testplanlægning, er det vigtigt at give tid til periodisk at afvikle de automatiserede testcases manuelt for at bevare kendskabet til, hvordan testen virker og for at verificere korrekt virkemåde så vel som for at kontrollere inddatas validitet og dækning.

7.2.3.5 Nøgleordsdrevet automatisering

Nøgleord (nogle gange omtalt som aktionsord) er mest, men ikke kun, brugt til at repræsentere højniveau forretningsinteraktioner med et system (f.eks. “fortryd ordre”). Hvert nøgleord anvendes typisk til at repræsentere et antal detaljerede interaktioner mellem en aktør og systemet under test. Sekvenser af nøgleord (inkl. relevant testdata) bruges til at specificere testcases.[Buwalda01]

I testautomatisering bliver et nøgleord implementeret som et eller flere eksekverbare testscripts. Værktøjer læser testcases skrevet som en sekvens af nøgleord, der kalder de rette testscripts, som implementerer nøgleordenes funktionalitet. Scripts er implementeret på en meget modulær måde for at give mulighed for nem mapning til specifikke nøgleord. Programmeringsevner er nødvendige for at implementere disse modulære scripts.

De primære fordele ved nøgleordsdrevet testautomatisering er:

- Nøgleord, der relaterer til en bestemt applikation eller forretningsdomæne kan defineres af domæneeksperter. Dette kan gøre arbejdet med testcasespecifikationen mere effektivt

- En person med fortrinsvis domæneekspertise kan have gavn af automatisk testcaseafvikling (når nøgleordene er blevet implementeret som scripts) uden at behøve at forstå den underliggende automatiseringskode
- Testcases skrevet med brug af nøgleord er lettere at vedligeholde, fordi det er mindre sandsynligt, at de skal ændres, hvis der ændres i softwaren under test
- Testcasespecifikationer er uafhængige af deres implementering. Nøgleordene kan implementeres ved brug af forskellige scriptsprog og -værktøjer.

Automatiseringsscripts (selve automatiseringskoden), der bruger nøgleords-/aktionsordsinformationen er ofte skrevet af udviklere eller tekniske testanalytikere, mens testanalytikeren normalt opretter og vedligeholder nøgleords-/aktionsdata. Mens nøgleordsdreven automatisering normalt forekommer i systemtestfasene, kan kodeudvikling begynde så tidligt som i integrationsfaserne. I et iterativt miljø er udvikling af testautomatisering en kontinuert proces.

7.2.3.6 Årsager til at automatiseringsindsatsen fejler

Når input-nøgleord og data er oprettet, overtager testanalytikeren normalt ansvaret for at afvikle de nøgleordsdrevne testcases og for at analysere enhver afvigelse, der måtte vise sig. Når man opdager en uregelmæssighed må testanalytikeren undersøge årsagen til afvigelsen for at fastslå om det skyldes nøgleord, input data, selve automatiseringsscriptet eller applikationen, der testes. Normalt vil det første skridt i fejlfindingen være at afvikle den samme test med de samme data manuelt for at se, om afvigelsen skyldes selve applikationen. Hvis dette ikke viser en afvigelse, bør testanalytikeren reviewe den sekvens af tests der gik forud for afvigelsen for at fastlægge, om problemet indtraf i et tidligere trin (måske ved at det producerede ukorrekte data), mens problemet ikke viste sig før senere i afviklingen. Hvis testanalytikeren ikke selv kan afgøre årsagen til en afvigelse, bør fejlfindingsinformationen overdrages til en teknisk testanalytiker eller en udvikler til videre analyse.

Projekter for automatisering af testafvikling opfylder ofte ikke deres mål. Dette kan være på grund af manglende fleksibilitet ved testværktøjet, utilstrækkeligt kendskab til programmering i testteamet eller urealistiske forventninger til hvilke problemer, der kan løses med automatiseret test. Al automatiseret testafvikling kræver ledelse, flid, evner og opmærksomhed, præcis som alle andre opgaver i et softwareudviklingsprojekt. Opgaven kræver en holdbar arkitektur, god designpraktik, konfigurationsstyring og god kode. De automatiserede testscripts indeholder sandsynligvis defekter og skal derfor testes, lige som de måske skal rettes af hensyn til performance. Man må bedømme brugbarheden af værktøjerne både for udvikleren og for brugerne. Det kan være nødvendigt at tilføje en grænseflade, der kan give testeren adgang til en logisk organisering af testcases.

8 Referencer

8.1 Standarder

- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements og Evaluation (SQuaRE)
  Kapitel 1 og 4
- [ISO9126] ISO/IEC 9126-1:2001, Software Engineering - Software Product Quality,
  Kapitel 1 og 4
- [RTCA DO-178B/ED-12B]: Software Considerations in Airborne Systems og Equipment Certification, RTCA/EUROCAE ED12B.1992, Kapitel 1

8.2 ISTQB dokumenter

- [ISTQB_AL_OVIEW] ISTQB Advanced Level Overview, Version 1.0
- [ISTQB_ALTM_SYL] ISTQB Advanced Level Test Manager Syllabus, Version 1.0
- [ISTQB_ALTTA_SYL] ISTQB Advanced Level Technical Test Analyst Syllabus, Version 1.0
- [ISTQB_FL_SYL] ISTQB Foundation Level Syllabus, Version 2011
- [ISTQB_GLOSSARY] Standard glossary of terms used in Software Test, Version 2.2, 2012

8.3 Bøger

[Bath08] Graham Bath, Judy McKay, “The Software Test Engineer's Handbook”, Rocky Nook, 2008, ISBN 978-1-933952-24-6

[Beizer95] Boris Beizer, “Black-box Test”, John Wiley & Sons, 1995, ISBN 0-471-12094-4

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

[Black07]: Rex Black, “Pragmatic Software Test”, John Wiley og Sons, 2007, ISBN 978-0-470-12790-2

[Buwalda01]: Hans Buwalda, “Integrated Test Design og Automation”, Addison-Wesley Longman, 2001, ISBN 0-201-73725-6

[Cohn04]: Mike Cohn, “User Stories Applied: For Agile Software Development”, Addison-Wesley Professional, 2004, ISBN 0-321-20568-5

[Copeland03]: Lee Copeland, “A Practitioner's Guide to Software Test Design”, Artech House, 2003, ISBN 1-58053-791-X

[Craig02]: Rick David Craig, Stefan P. Jaskiel, “Systematic Software Test”, Artech House, 2002, ISBN 1-580-53508-9

[Gerrard02]: Paul Gerrard, Neil Thompson, “Risk-based e-business Test”, Artech House, 2002, ISBN 1-580-53314-0

[Gilb93]: Tom Gilb, Graham Dorothy, “Software Inspection”, Addison-Wesley, 1993, ISBN 0-201-63181-4

[Graham07]: Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black “Foundations of Software Test”, Thomson Learning, 2007, ISBN 978-1-84480-355-2

[Grochmann94]: M. Grochmann (1994), Testcase design using Classification Trees, in: conference proceedings STAR 1994

[Koomen06]: Tim Koomen, Leo van der Aalst, Bart Broekman, Michiel Vroon "TMap NEXT, for result driven test", UTN Publishers, 2006, ISBN 90-72194-80-2

[Myers79]: Glenford J. Myers, “The Art of Software Test”, John Wiley & Sons, 1979, ISBN 0-471-46912-2

[Splaine01]: Steven Splaine, Stefan P. Jaskiel, “The Web-Test Handbook”, STQE Publishing, 2001, ISBN 0-970-43630-0

[vanVeenendaal12]: Erik van Veenendaal, “Practical risk-based test – The PRISMA approach”, UTN Publishers, The Netherlands, 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 Test”, Addison-Wesley, 2009, ISBN 0-321-63641-4

8.4 Andre referencer

De følgende referencer henviser til til information, der er tilgængelig på Internettet eller andre steder. Selv om disse referencer var gældende på udgivelsestidspunktet for dette pensum for Advanced Test Analyst, kan ISTQB ikke holdes ansvarlig, hvis referencerne ikke længere er tilgængelige.

Kapitel 3

- Czerwonka, Jacek: www.pairwise.org
- Defektaksonomi: www.testeducation.org/a/bsct2.pdf
- Eksempel på defekttaksonomi baseret på Boris Beizer's arbejde: inet.uni2.dk/~vinter/bugtaxst.doc
- Godt overblik over forskellige taksonomier: testeducation.org/a/bugtax.pdf
- Heuristic Risk-Based Test By James Bach
- Fra “Exploratory & Risk-Based Test (2004) www.testeducation.org"
- Exploring Exploratory Test , Cem Kaner og Andy Tikam , 2003
- Pettichord, Bret, “An Exploratory Test Workshop Report”,
- www.testcraft.com/exploratorypettichord

Kapitel 4

- www.teststandards.co.uk