Foundation Level Syllabus
Certified Tester

Version 2018 v3.1

International Software Testing Qualification Board

Dansk udgave v1.4

Udgivet: December, 2020

Besked om ophavsrettigheder

Revisionshistorie

Revisionshistorik for Engelsk version

Version	Dato	Bemærkninger
ISEB v2.0	25. februar 1999	ISEB Foundation Level Certified Tester syllabus v2.0
ASQF v2.2	Juli 2003	ASQF, Foundation Level Certified Tester syllabus 2.2 "Lehrplan Grundlagen des Software-testens"
ISTQB® 2005	1. juli, 2005	ISTQB® CTFL Certified Tester Foundation Level syllabus
ISTQB® 2007	1. maj, 2007	ISTQB® CTFL Certified Tester Foundation Level Certified Tester syllabus vedligeholdelsesrelease
ISTQB® 2010	30. marts, 2010	ISTQB® CTFL Certified Tester Foundation Level syllabus vedligeholdelsesrelease – se releasesnoter
ISTQB® 2011	1. april, 2011	ISTQB® CTFL Certified Tester Foundation Level 2011 syllabus vedligeholdelsesrelease – se releasesnoter
ISTQB® 2018	27. april, 2018	ISTQB® CTFL Certified Tester Foundation Level 2018 syllabus
ISTQB® 2018 v3.1	11. november 2019	ISTQB® CTFL Certified Tester Foundation Level 2018 syllabus mindre opdatering

Revisionshistorik for Dansk oversættelse af ISTQB® CTFL 2018

Version	Dato	Bemærkninger
1.0	Marts, 2019	ISTQB® CTFL Certified Tester Foundation Level 2018 syllabus, dansk oversættelse
1.1	September, 2019	Mindre rettelser af på teksturelt plan
1.2	Maj, 2020	ISTQB® CTFL Certified Tester Foundation Level 2018 v3.1 syllabus, dansk oversættelse
1.3	August, 2020	Rettelse af referencer der forkert refererede til afsnit 0
1.4	December, 2020	Rettekse af termerne: ’systemegenskab’ og ’kvalitetskarakteristik’ til korrekt ’kvalitetsegenskab’

Indholdsfortegnelse

Anerkendelser

Dette dokument blev formelt frigivet af ISTQB® General Assemble (11. november 2019).

ISTQB® CTFL Foundation 2018 v3.1 blev produceret af et team under Internation Software Testing Qualification Board: Klaus Olsen (chair), Meile Posthuma og Stephanie Ulrich.

ISTQB® CTFL Foundation 2018 blev produceret af et team under International Software Testing Qualifications Board: Klaus Olsen (formand), Tauhida Parveen (næstformand), Rex Black (projectmanager), Debra Friedenberg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh og Eshraka Zakaria.

Teamet vil gerne takker Rex Black og Dorothy Graham for deres tekniske editering, til review teamet og krydsreview teamet, samt til lokale ISTQB® medlemmer for deres forslag og input.

Følgende personer deltog i review og kommentering på denne syllabus og for at stemning den igennem: Tom Adams, Tobias Ahlgren, Xu Aiguo, Chris Van Bael, Katalin Balla, Graham Bath, Gualtiero Bazzana, Arne Becher, Veronica Belcher, Lars Hilmar Bjørstrup, Ralf Bongard, Armin Born, Robert Bornelind, Mette Bruhn-Pedersen, Geza Bujdoso, Earl Burba, Filipe Carlos, Young Jae Choi, Greg Collina, Alessandro Collino, Cui Zhe, Taz Daughtrey, Matthias Daigl, Wim Decoutere, Frans Dijkman, Klaudia Dussa-Zieger, Yonit Elbaz, Ofer Feldman, Mark Fewster, Florian Fieber, David Frei, Debra Friedenberg, Conrad Fujimoto, Pooja Gautam, Thorsten Geiselhart, Chen Geng, Christian Alexander Graf, Dorothy Graham, Michel Grandjean, Richard Green, Attila Gyuri, Jon Hagar, Kobi Halperin, Matthias Hamburg, Zsolt Hargitai, Satoshi Hasegawa, Berit Hatten, Wang Hongwei, Tamás Horváth, Leanne Howard, Chinthaka Indikadahena, J. Jayapradeep, Kari Kakkonen, Gábor Kapros, Beata Karpinska, Karl Kemminger, Kwanho Kim, Seonjoon Kim, Cecilia Kjellman, Johan Klintin, Corne Kruger, Gerard Kruijff, Peter Kunit, Hyeyong Kwon, Bruno Legeard, Thomas Letzkus, Alon Linetzki, Balder Lingegård, Tilo Linz, Hongbiao Liu, Claire Lohr, Ine Lutterman, Marek Majernik, Rik Marselis, Romanos Matthaios, Judy McKay, Fergus McLachlan, Dénes Medzihradszky, Stefan Merkel, Armin Metzger, Don Mills, Gary Mogyorodi, Ninna Morin, Ingvar Nordström, Adam Novak, Avi Ofer, Magnus C Ohlsson, Joel Oliviera, Monika Stocklein Olsen, Kenji Onishi, Francisca Cano Ortiz, Gitte Ottosen, Tuula Pääkkönen, Ana Paiva, Tal Pe'er, Helmut Pichler, Michaël Pilaeten, Horst Pohlmann, Andrew Pollner, Meile Posthuma, Vitalijs Puiso, Salvatore Reale, Stuart Reid, Ralf Reissing, Shark Ren, Miroslav Renda, Randy Rice, Adam Roman, Jan Sabak, Hans Schaefer, Ina Schieferdecker, Franz Schiller, Jianxiong Shen, Klaus Skafte, Mike Smith, Cristina Sobrero, Marco Sogliani, Murian Song, Emilio Soresi, Helder Sousa, Michael Sowers, Michael Stahl, Lucjan Stapp, Li Suyuan, Toby Thompson, Steve Toms, Sagi Traybel, Sabine Uhde, Stephanie Ulrich, Philippos Vakalakis, Erik van Veenendaal, Marianne Vesterdal, Ernst von Düring, Salinda Wickramasinghe, Marie Walsh, Søren Wassard, Hans Weiberg, Paul Weymouth, Hyungjin Yoon, John Young, Surong Yuan, Ester Zabar og Karolina Zmitrowicz.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2018): Klaus Olsen (formand), Tauhida Parveen (næstformand), Rex Black (projectmanager), Dani Almog, Debra Friedenberg, Rashed Karim, Johan Klintin, Vipul Kocher, Corne Kruger, Sunny Kwon, Judy McKay, Thomas Müller, Igal Levi, Ebbe Munk, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, Eshraka Zakaria og Stevan Zivanovic. Teamet vil gerne takker review gruppen og lokale ISTQB® medlemmer for deres input.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2011): Thomas Müller (formand), Debra Friedenberg. Teamet vil gerne takker review gruppen (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier Hans Schaefer, Stephanie Ulrich og Erik van Veenendaal) og alle lokale ISTQB® medlemmer for deres input.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2010): Thomas Müller (formand), Rahul Verma, Martin Klonk og Armin Beer. Teamet vil gerne takker review gruppen (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams og Erik van Veenendaal) og alle lokale ISTQB® medlemmer for deres input.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2007): Thomas Müller (formand), Dorothy Graham, Debra Friedenberg og Erik van Veenendaal. Teamet vil gerne takker review gruppen (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson og Wonil Kwon) og alle lokale ISTQB® medlemmer for deres input.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2005): Thomas Müller (formand), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson og Erik van Veenendaal. Teamet vil gerne takker review gruppen og alle lokale ISTQB® medlemmer for deres input.

Oversættelse

DSTB vil gerne takke Bent Hemmingsen for den danske oversættelse af ISTQB® Certified Tester 2018 og Klaus Skafte for oversættelse af version 2018 v3.1.

DSTB vil gerne takker følgende personer for at hjælpe med oversættelse og review af Begrebslisten i forbindelse med Certified Tester 2018 og v3.1 syllabus: Klaus Skafte (Projektleder), Anja P. Nielsen, Birgitte Frisner, Ebbe Munk, Egil Boisen, Fehmeed Ijaz, Gitte Ottosen, Henrik Skjærbæk, Jane M. Nash, Jesper Petersen, Knud Hangaard, Mandana Bahar, Maria Jensen, Martin N. Poulsen, Mette Bruhn-Pedersen, Morten Keblovszki Fredens, Ole Chr. Hansen, Søren Wassard og Tina M. Knudsen

DSTB vil gerne takker følgende personer for review af den danske Certified Tester 2018 og v3.1 oversættelse: Klaus Skafte (Projektleder), Ebbe Munk, Gitte Ottosen, Ole Chr. Hansen og Søren Wassard

0. Introduktion

0.1 Formålet med denne syllabus

Denne syllabus udgør grundlaget for International Software Testing Qualification at the Foundation Level (internationalt kvalificerende softwaretest på grundniveau). ISTQB® kan udlevere denne syllabus i følgende tilfælde:

1. Til lokale ISTQB®-organisationer, så det kan blive oversat til lokale sprog og til akkreditering af undervisere. De lokale organisationer kan tilpasse syllabus efter deres lokale, sproglige behov og tilføje henvisninger, så syllabus kan tilpasses de lokale publikationer
2. Til certificeringsorganer, der så kan skrive spørgsmål på eget, lokale sprog, som tilpasses læringsmålene i denne syllabus
3. Til udbydere af undervisning, at producere kursusmaterialer og at finde frem til de passende undervisningsmetoder
4. Til certificeringskandidater og forberede certificeringseksamen (enten som en del af et trænings-kursus eller selvstændigt)
5. Til det internationale software- og systemteknikermiljø, at udvikle software- og systemtests-professionen og at udgøre grundlaget for bøger og artikler

ISTQB® kan tillade, at andre organisationer kan anvende denne syllabus til andre formål, hvis de på forhånd får skriftlig akkreditering fra ISTQB®.

0.2 Certificeret tester på Foundation Level i softwaretest

Foundation Level kvalifikationen beregnet til alle, der arbejder med softwaretest. Dette omfatter roller som testere, testanalytikere, testteknikere, testkonsulenter, testmanagers, user acceptance-testere og softwareudviklere. Denne Foundation Level kvalificering er beregnet til enhver, der ønsker en grundlæggende forståelse for softwaretest, såsom product owners, projektledere, kvalitetschefer, softwareudviklingsledere, forretningsanalytikere, IT-direktører og ledelseskonsulenter. Med en certificering som tester på Foundation Level kan man senere opgradere sine kvalifikationer til et højere niveau af certificeringer.

ISTQB® CTFL Foundation Level Overview 2018 er et separat dokument med følgende oplysninger:

- Forretningsmæssige gevinster ved studieplanen
- En matrice, der viser sporbarhed mellem forretningens resultater og læringsmålene
- Resumé af denne syllabus

ISTQB® CTFL Foundation Level Overview 2018 er stadig retvisende for denne version af syllabussen.

0.3 Eksamensrelevante læringsmål og kognitive vidensniveauer

Læringsmålene understøtter forretningens resultater og anvendes til at udarbejde eksaminer for Certified Tester Foundation Level.

Generelt er alt indhold i denne syllabus eksamensrelevant på K1-niveau, bortset fra Indledning og Bilag. Det vil sige, at kandidaten kan forventes at kunne genkende, huske eller erindre nøgleord eller koncepter fra de seks kapitler. De specifikke læringsmåls vidensniveauer vises i begyndelsen af hvert kapitel og klassificeres som følgende:

- K1: Husk
- K2: Forstå
- K3: Anvend

Yderligere detaljer og eksempler på læringsmål beskrives i Bilag B.

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

0.4 Certificeringseksamen på Foundation Level

Certificeringseksamen på Foundation Level er baseret på denne syllabus. Svar på eksamensspørgsmålene kan kræve anvendelse af materialer, der bygger på mere end et afsnit i denne syllabus. Alle afsnit i syllabus er eksamensrelevante, bortset fra Indledning og Bilag. Der er henvist til standarder, bøger og andre ISTQB®-studieplaner, men sådanne kilder er ikke eksamensrelevante ud over hvad er er opsummeret her.

Eksamen foregår som multiple choice. Der er 40 spørgsmål, og for at bestå eksamen skal mindst 65% af spørgsmålene (dvs. 26 spørgsmål) besvares korrekt.

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

0.5 Akkreditering

Lokale ISTQB®-medlemmer kan akkreditere undervisere, hvis deres undervisningsmateriale følger denne syllabus. Undervisere kan få retningslinjer for bestyrelsen eller akkrediteringsorganet. Et akkrediteret kursus skal følge denne syllabus og kan afholde ISTQB®-eksaminer som en del af kurset.

0.6 Detaljeniveau

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

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

Syllabus indhold er ikke en beskrivelse af hele vidensområdet for softwaretest; det gengiver blot det detaljeniveau, der skal dækkes i Foundation Level træningskurser. Syllabus fokuserer på de testkoncepter og -teknikker, som kan overføres til alle softwareprojekter, herunder agile projekter. Denne syllabus indeholder ikke specifikke læringsmål for nogen særlig livscyklus eller nogen særlig metode for softwareudvikling, men i syllabus beskrives f.eks., hvordan disse koncepter fungerer i agile projekter, andre typer af iterative og inkrementelle livscyklusser og sekventielle livscyklusser.

0.7 Syllabus indhold

Der er seks kapitler med eksamensrelevant indhold. Overskriften for hvert kapitel nævner den forventede undervisningstid for kapitlet; der angives ikke tid under kapitelniveau. For de akkrediterede kurser kræver syllabus som minimum 16,75 timers undervisning, der fordeles over de seks kapitler på følgende måde:

- Kapitel 1: 175 minutter Grundlæggende test
- Kapitel 2: 100 minutter Test igennem hele softwareudviklingslivscyklussen
- Kapitel 3: 135 minutter Statisk test
- Kapitel 4: 330 minutter Testteknikker
- Kapitel 5: 225 minutter Teststyring
- Kapitel 6: 40 minutter Værktøjsstøtte til test

1. Grundlæggende test — 175 minutter

Nøgleord

afvigelse, debugging, defekt, dækning, fejl, kvalitet, kvalitetssikring, rodårsag, sporbarhed, test, testafsluning, testafvikling, testanalyse, testcase, testdata, testdesign, testfokus, testformål, testgrundlag, testimplementering, testkontrol, testobjekt, testorakel, testovervågning, testplanlægning, testprocedure, testproces, testsuite, testware, validering, verifikation

Læringsmål for Grundlæggende test:

1.1 Hvad vil det sige at teste?

FL-1.1.1 (K1) Identificer typiske målsætninger for testen
FL-1.1.2 (K2) Forklar forskellene på test og debugging

1.2 Hvorfor er test nødvendig?

FL-1.2.1 (K2) Giv eksempler på, hvorfor test er nødvendig
FL-1.2.2 (K2) Beskriv forholdet mellem test og kvalitetssikring, og giv os eksempler på, hvordan tests bidrager til højere kvalitet
FL-1.2.3 (K2) Forklar forskellen mellem fejl, defekt og afvigelser
FL-1.2.4 (K2) Forklar forskellen mellem defektens rodårsag og dens effekter

1.3 Syv testprincipper

FL-1.3.1 (K2) Forklar de syv testprincipper

1.4 Testprocessen

FL-1.4.1 (K2) Forklar kontekstens indvirkning på testprocessen
FL-1.4.2 (K2) Beskriv testaktiviteterne og de respektive opgaver inden for testprocessen
FL-1.4.3 (K2) Redegør for forskellene mellem de arbejdsprodukter, der understøtter testprocessen
FL-1.4.4 (K2) Forklar værdien af at opretholde sporbarheden mellem testgrundlaget og testens arbejdsprodukter

1.5 Testens psykologi

FL-1.5.1 (K1) Identificer de psykologiske faktorer, der har indvirkning på testens succes
FL-1.5.2 (K2) Forklar forskellen mellem det mindset, der er nødvendigt ved testaktiviteterne, og det mindset, der er nødvendigt ved udviklingsaktiviteter

1.1 Hvad vil det sige at teste?

Softwaresystemer er en integreret del af det moderne menneskes liv. De bliver brugt overalt i hverdagen fra banker til biler. De fleste mennesker kender til software, der ikke fungerede som forventet. Software, der ikke fungerer, som det skal, kan lede til mange problemer, f.eks. tab af tid, penge og omdømme, og kan i værste fald lede til personskader eller dødsfald. Softwaretest er en måde, hvor man kan vurdere softwarens kvalitet og minimere risikoen for softwareafvigelse i drift.

Det er en almindelig misforståelse, at test kun består i at afvikle tests, dvs. at køre softwaren og gennemgå resultaterne. Som beskrevet i afsnit 1.4 er en softwaretest en proces, der indeholder mange forskellige aktiviteter; afvikling af testen (og at gennemgå resultaterne) er blot en af disse aktiviteter. Testprocessen består f.eks. også af planlægning af testen, analyse, design og implementering, rapportering af fremdrift og resultater og det at evaluere kvaliteten af testobjektet.

Nogle tests omfatter afvikling af softwaren i komponenten eller systemet; det kaldes dynamisk test. Andre former for tests inkluderer ikke afvikling af software i komponent eller system, det kaldes en statisk test. En test kan også bestå i en gennemgang af arbejdsproduktet, f.eks. krav, userstories eller kildekode.

En anden almindelig misforståelse at test udelukkende fokuserer på verifikation af krav, userstories og andre specifikationer. Selvom test omfatter en verifikation af, om et system lever op til de specificerede krav, indgår der også validering, hvilket vil sige, at man undersøger, om systemet lever op til brugerens og andre interessenters behov i driftsmiljø(er).

Testaktiviteter planlægges og udføres forskelligt i forskellige livscyklusser (se afsnit 2.1).

1.1.1 Testens typiske målsætninger

For ethvert givet projekt kan testens målsætninger omfatte:

- At forebygge defekter ved at evaluere arbejdsprodukter som f.eks. krav, userstories, design og kode
- At bedømme, om de specificerede krav er opfyldt
- At godkende, at et testobjekt er færdiggjort og fungerer i henhold til brugernes og andre interessenters forventninger
- At opbygge tillid til testobjektets kvalitetsniveau
- At finde defekter og afvigelser og derved reducerer risikoen for dårlig kvalitet i softwaren
- At levere tilstrækkelige oplysninger til interessenterne for at give dem mulighed for at foretage beslutninger på et oplyst grundlag, især om testobjektets kvalitetsniveau
- At følge kontrakter, love, regler og standarder, og/eller at sikre, at testobjektet lever op til sådanne krav eller standarder

Testens målsætninger kan variere afhængigt af den kontekst, komponenten eller systemet testes i, af testniveauet og af softwareudviklingens livscyklusmodel. Disse målsætninger kan f.eks. omfatte:

- I løbet af komponenttesten kan det være en målsætning at finde så mange afvigelser som muligt, så underliggende defekter kan identificeres og udbedres tidligt. En anden målsætning kan være at øge kodedækningen i komponenttesten
- I løbet af accepttesten kan det være en målsætning at bekræfte, at systemet fungerer som forventet og lever op til kravene. En anden målsætning kan være at bedømme risici ved leverance af systemet på et givet tidspunkt og at formidle disse

1.1.2 Test og debugging

Test er forskellig fra debugging. Afvikling af test kan vise afvigelser, der er forårsaget af defekter i softwaren. Debugging er en udviklingsaktivitet, hvor man finder, analyserer og afhjælper sådanne defekter. Yderligere gentest undersøger, om tiltagene har afhjulpet defekterne. I nogle tilfælde er testerne ansvarlige for den tidlige test og den endeligt bekræftende test, mens udviklerne udfører fejlfinding og dertil hørende test af komponenter og komponentintegrationer (continuous integration). I agil udvikling og tilsvarende livscyklusser kan testere inddrages i debugging og komponenttesten.

ISO-standard (ISO/IEC/IEEE 29119-1) indeholder yderligere oplysninger om koncepter for softwaretest.

1.2 Hvorfor er test nødvendig?

Omhyggelig test af komponenter og systemer og den tilhørende dokumentation kan hjælpe med at reducere risikoen for afvigelser, som kan opstå under drift. Hvis defekterne bliver fundet og udbedret, bidrager det til komponentens eller systemets kvalitet. Desuden kan softwaretest også være nødvendig for at opfylde kontraktkrav, lovkrav og industristandarder.

1.2.1 Testens bidrag til vellykkede projekter

Igennem computerens historie har det været almindeligt, at software og systemer er blevet leveret til drift med defekter, der senere har forårsaget afvigelser eller at produktet på anden vis ikke har levet op til interessenternes behov. Ved hjælp af passende testteknikker kan hyppigheden af sådanne problematiske leverancer reduceres, hvis teknikkerne anvendes med det rette niveau af testekspertise, i passende test-niveauer og i passende faser i softwareudviklingens livscyklus. Eksempler:

- Hvis testere er involveret i gennemgangen af krav og raffinering af userstories kan hjælpe med at spore defekter i disse arbejdsprodukter. Identifikation og afhjælpning af defekter reducerer risikoen for udvikling af ukorrekt og ikke-testbar funktioner
- Hvis testere arbejder tæt sammen med systemarkitekter, mens systemet bliver designet, kan øge begge parters forståelse for projektet, og hvordan man bedst tester det. Den øgede forståelse reducerer risikoen for grundlæggende fejl i designet og gør det muligt at finde fejl på et tidligt stadie
- Hvis testere arbejder tæt sammen med udviklere, mens koden udvikles, kan øge begge parters forståelse af koden, og hvordan man bedst tester den. Det kan reducere risikoen for at der opstår defekter i kode og tests
- Hvis testere verificerer og validerer softwaren inden releasen kan bidrage til at afvigelser, der ellers ville være blevet overset, bliver fundet, og det støtter processen med at finde afvigelser. Det øger sandsynligheden for, at softwaren opfylder interessenternes behov og lever op til kravene

Ud over disse eksempler bidrager opnåelsen af de definerede målsætninger for testen (se afsnit 1.1.1) til vellykket softwareudvikling og -drift.

1.2.2 Kvalitetssikring og test

Mange anvender begrebet kvalitetssikring (QA) i stedet for test, men kvalitetssikring og test ikke det samme, selvom de er beslægtede begreber. Det overordnede koncept "kvalitetsstyring" binder dem sammen. Kvalitetsstyring omfatter alle aktiviteter, der styrer eller kontrollerer en organisation i henhold til kvalitetsbegrebet. Blandt de andre aktiviteter, som kvalitetsstyring indebærer, indgår både kvalitetssikring og kvalitetskontrol. Kvalitetssikring fokuserer primært på at følge de korrekte processer for at skabe tillid til, at de rette kvalitetsniveauer opnås. Hvis processen er udført korrekt, har arbejdsproduktet, der er skabt ud fra disse processer, generelt en højere kvalitet, hvilket bidrager til forebyggelse af defekter. Desuden kan anvendelse af rodårsagsanalyser til at spore og fjerne årsager til defekter sammen med korrekt anvendelse af fund fra retrospektive møder forbedre processerne og er dermed vigtige for effektiv kvalitetssikring.

Kvalitetsstyring indebærer forskellige aktiviteter, herunder testaktiviteter, der støtter opnåelse af passende kvalitetsniveauer. Testaktiviteter er en del af den overordnede softwareudvikling eller vedligeholdelsesproces. Da kvalitetssikring drejer sig om korrekt udførsel af hele processen, sørger kvalitetssikringen for at testene bliver udført korrekt. Som beskrevet i afsnit 1.1.1 og 1.2.1 bidrager testen til kvaliteten på forskellige måder.

1.2.3 Fejl, defekter og afvigelser

En person kan begå en fejl (fejltagelse), der kan lede til introduktion af en defekt (afvigelse eller bug) i koden eller relateret arbejdsprodukt. En fejl, der introducerer en defekt i et arbejdsprodukt, kan udløse en anden fejl, der introducerer en defekt i et relateret arbejdsprodukt. F.eks. kan en fejl i kravene lede til en defekt i kravene, hvilket så resulterer i en programmeringsfejl, der leder til defekter i koden.

Defekter i koden vil ikke altid udløse afvigelser. F.eks. kræver nogle defekter meget specifikke input eller forudsætninger for at udløse en afvigelse, hvilket kan være en sjælden eller umulig forekomst.

Fejl kan opstå af mange årsager, så som:

- Tidspres
- Menneskelig fejlbarlighed
- Uerfarne eller utilstrækkeligt uddannede projektdeltagere
- Fejlkommunikation mellem projektdeltagere, blandt andet om krav og design
- Kodens kompleksitet, design og arkitektur, de underliggende problemer, der skal løses, og/eller de anvendte teknologier
- Misforståelser om grænseflader for intra-systemer og inter-systemer, særligt hvis der er mange interaktioner
- Nye, ukendte teknologier

Ud over afvigelser forårsaget af defekter i koden, kan afvigelser også være forårsaget af miljømæssige omstændigheder. F.eks. kan radioaktivitet, elektromagnetiske felter og forurening forårsage defekter i firmware eller påvirke afviklingen af softwaren ved at ændre hardwarens tilstand.

Ikke alle uventede resultater er afvigelser. Der kan opstå falsk positive resultater som resultat af afvigelser i den måde testen er blevet afviklet på, pga. defekter i testdataene, testmiljøet eller testapplikationer eller af en anden årsag. Den omvendte situation kan også forekomme, hvor tilsvarende fejl eller defekter kan lede til falsk negative resultater. Falske negativer er tests, der ikke sporer defekter, som burde have været opdaget; falske positiver rapporteres som defekter uden at være det.

1.2.4 Defekter, rodårsager og effekter

Defekternes rodårsager er de tidligste handlinger, som har bidraget til at danne defekterne. Defekter kan blive analyseret for at identificere rodårsager for at mindske tilfælde af lignende defekter i fremtiden. Ved at fokusere på de mest betydende rodårsager, kan rodårsagsanalysen lede til procesforbedringer, der forhindrer et betydeligt antal fremtidige defekter i at opstå.

F.eks. kan der ske ukorrekt beregning af rente pga. en enkelt, ukorrekt kodelinje, der kan resultere i klager fra kunder. Den defekte kode blev skrevet til en userstory, der var tvetydig, pga. product owners misforståelse af, hvordan man skal udregne renter. Hvis en stor procentdel af defekterne findes i renteudregningen, og disse defekter udgør en rodårsag i lignende misforståelser, kan product owner trænes i emnet renteudregning for at mindske sådanne defekter i fremtiden.

I dette eksempel udgør kundens klage effekten. Den ukorrekte renteberegning er en afvigelse. Den ukorrekte udregning i koden er en defekt, og den opstod ud fra den oprindelige fejl, altså tvetydigheden i en userstory. Den oprindelige defekts rodårsag var product owners manglende viden, hvilket gjorde, at product owner begik en fejl, da hun skrev sin userstory. Rodårsagsanalysens proces gennemgås i ISTQB® CTEL-TM Certified

Tester Expert Level Test Management syllabus og ISTQB® CTEL-ITP Certified Tester Expert Level Improving the Test Process syllabus.

1.3 Syv testprincipper

I løbet af de sidste 50 år er der foreslået en række testprincipper der definerer de generelle retningslinjer, som anvendes i al test.

1. Test viser tilstedeværelse af defekter, ikke deres fravær

En test kan vise, at der findes defekter, men kan ikke bevise, at der ingen defekter er. En test kan mindske sandsynligheden for, at der stadig findes uopdagede defekter i softwaren, men selv hvis testen ikke finder defekter, er testen ikke et bevis for, at koden er korrekt.

2. Udtømmende test er umulig

Det er umuligt at teste alle kombinationer af input og forudsætninger, selv i trivielle tilfælde. I stedet for at forsøge at teste udtømmende, bør man anvende risikoanalyse, testteknikker og prioritering for at begrænse testens fokus.

3. Tidlig test sparer både tid og penge

For at finde defekter tidligt, bør både statiske og dynamiske tests påbegyndes så tidligt som muligt i softwareudviklingens livscyklus. Tidlig test kaldes også shift left. En test tidligt i softwareudviklingens livscyklus hjælper til at mindske eller undgå dyre ændringer (se afsnit 3.1)

4. Defekter samler sig i klynger

Et lille antal moduler indeholder normalt de fleste af de defekter som man finder i løbet af tests før releasen, eller er ansvarlige for de fleste driftsafvigelser. Det er vigtigt at medregne de forventede klynger af defekter og de observerede klynger af defekter i test eller drift i den risikoanalyse, der skal anvendes til at indsnævre testens fokus (som nævnt i princip 2).

5. Pas på pesticidparadokset

Hvis samme test gentages igen og igen vil den med tiden ikke finde nye defekter. For at finde nye defekter er der behov for at ændre de eksisterende tests og testdata, og at udarbejde nye tests. (Tests, der ikke længere er i stand til at finde defekter, ligesom pesticider med tiden mister evnen til at dræbe insekter.) I nogle tilfælde, f.eks. ved automatiseret regressionstest, har pesticidparadokset et fordelagtigt udfald, hvilket er det relativt lave antal regressionsdefekter.

6. Test er afhængig af konteksten

En test udføres forskelligt alt efter konteksten. F.eks. skal sikkerhedskritisk industriel styringssoftware testes anderledes end en e-commerce app. Som et andet eksempel skal test afvikles på én måde i et agilt projekt og på en anden måde i et sekventielt softwareudvilkingscyklusprojekt (se afsnit 2.1)

7. Fravær-af-fejl-fejltagelsen

Nogle organisationer forventer, at testere kan gennemføre alle mulige tests og finde alle tilstedeværende defekter, men ud fra henholdsvis princip 2 og 1 ved vi, at det ikke er muligt. Desuden er det fejlagtigt at forvente, at blot man finder og udbedrer et stort antal af defekter, så sikrer det, at systemet er vellykket. F.eks. kan man vha. grundig test af alle specificerede krav og at udbedre alle defekter stadig fremstille et system, der er svært at anvende, som ikke lever op til brugernes behov og forventninger eller som er underlegent til sammenligning med konkurrerende systemer.

Se Myers 2011, Kaner 2002 og Weinberg 2008 for eksempler på disse og andre testprincipper.

1.4 Testprocessen

Der findes ikke en universel testproces for software, men der findes et generelt sæt af testaktiviteter, foruden hvilke testen har en ringere chance for at opnå sine forudbestemte målsætninger. Dette sæt af testaktiviteter udgør en testproces. Hvad der er den korrekte softwaretestproces i en given situation, afhænger af mange faktorer. Hvilke testaktiviteter, som testprocessen indeholder, hvordan de implementeres, og hvornår de finder sted, kan diskuteres inden for organisationens teststrategi.

1.4.1 Testprocessen i kontekst

Kontekstuelle faktorer, der påvirker organisationens testproces, gælder bl.a.:

- Modeller og projektmetodologier, der anvendes i softwareudviklingens livscyklus
- Testniveauer og testtyper der overvejes anvendt
- Produkt og projektrisici
- Forretningsdomæne
- Operationelle begrænsninger omfatter, men er ikke begrænset til:
  - Budgetter og resurser
  - Tidshorisonter
  - Kompleksitet
  - Kontraktlige og lovmæssige krav
- Organisatoriske politikker og praksisser
- Påkrævede interne og eksterne standarder

De følgende afsnit beskriver de generelle sider ved de organisatoriske testprocesser i forhold til det følgende:

- Testaktiviteter og -opgaver
- Testens arbejdsprodukter
- Sporbarheden mellem testgrundlaget og testarbejdsprodukter

Det er meget anvendeligt, hvis testgrundlaget (for alle de niveauer eller typer af test, der overvejes) har målbare dækningskriterier specificeret. Dækningskriteriet kan fungere effektivt som key performance indikatorer (KPI'er) til at styre de aktiviteter, der viser resultaterne af softwaretestens målsætninger (se afsnit 1.1.1).

F.eks. kan testgrundlaget til en app indeholde en liste af krav og en liste over understøttede mobilenheder. Hvert krav udgør et element af testgrundlaget. Hver understøttede enhed udgør også et element af test-grundlaget. Dækningskriteriet kan indeholde mindst en testcase for hvert af testgrundlagets elementer. Når først testen er afviklet, kan resultaterne fortælle interessenterne, hvorvidt produktet lever op til de specificerede krav og om man har observeret fejl på de understøttede enheder.

ISO-standard (ISO/IEC/IEEE 29119-2) indeholder yderligere oplysninger vedrørende testprocesserne.

1.4.2 Testaktiviteter og opgaver

En testproces indeholder følgende hovedgrupper af aktiviteter:

- Testplanlægning
- Testovervågning og -kontrol
- Testanalyse
- Testdesign

- Testimplementering
- Testafvikling
- Testafslutning

Hver hovedgruppe af aktiviteter er sammensat af aktiviteter, som er beskrevet i underafsnittene nedenfor. Aktiviteterne kan bestå af adskillige individuelle opgaver, som kan variere fra et projekt eller release til det næste.

Selv om mange af disse aktivitetshovedgrupper kan forekomme i logiske sekvenser, implementeres de ofte iterativt. F.eks. involverer agil udvikling små iterationer af software design, builds og test, der forekommer på kontinuerlig basis og bliver understøttet af igangværende planlægning. Derfor forekommer testaktiviteter også på iterativ og kontinuerlig basis inden for denne software udviklingstilgang. Selv inden for sekventiel softwareudvikling, vil den trinvise logiske aktivitetssekvens involvere overlap, kombinationer, samtidige forekomster eller udeladelser, så det er generelt nødvendigt, at man tilpasser disse hovedgruppeaktiviteter til både systemets og projektets kontekster.

Testplanlægning
Planlægning af testen involverer aktiviteter, der definerer testens målsætninger og den fremgangsmåde, hvorpå testens målsætninger skal opnås inden for de rammer, der fremsættes af konteksten (f.eks. udvælgelse af passende testteknikker og opgaver og sammensætning af tidsplanen for testen, så den overholder tidsfristen). Testplanen kan revideres på grundlag af feedback fra overvågnings- og kontrolaktiviteter. Testplanlægning bliver yderligere forklaret i afsnit 5.2.

Testovervågning og -kontrol
Testovervågning omfatter den fortløbende sammenligning af den faktiske fremdrift over for den planlagte fremdrift ved at anvende de testovervågningsmetrikker, som står beskrevet i testplanen. Testkontrol omfatter det at foretage de nødvendige handlinger for at opnå testplanens målsætninger (der kan blive opdateret løbende). Overvågning og styring af testen understøttes af evalueringen af slutkriterierne, hvilket henvises til som Definition of Done i nogle softwareudviklingslivscyklusmodeller (se ISTQB® CTFL-AT Certified Tester Foundation Level Agile Tester syllabus). F.eks. kan evalueringen af slutkriteriet for udførslen af testen som en del af den pågældende test omfatte:

- At tjekke testresultaterne og logs mod de specificerede dækningskriterier
- Vurdere komponentens eller systemkvalitetens niveau på grundlag af testresultater og loggen
- Afgøre om yderligere test er nødvendige (f.eks. hvis man med testene oprindeligt forsøgte at opnå et vist niveau af risikodækning for produktet, men at dette ikke lykkedes, og der derfor kræves, at yderligere test udfærdiges og udføres)

Testfremdrift i forhold til planen kommunikeres til interessenterne i testens fremdriftsrapport, inklusive afvigelser fra planen og oplysninger, der ligger til grund for at stoppe testen.

Testovervågning og -styring forklares yderligere i afsnit 5.3.

Testanalyse
I løbet af testanalysen bliver testgrundlaget analyseret for at identificere testbare egenskaber og definere de tilknyttede testbetingelser. Med andre ord afgør testanalysen "hvad man skal teste" i forhold til målbare dækningskriterier.

Testanalysen indebærer følgende større aktiviteter:

- Analyse af det testgrundlag, som er passende på det testniveau, der overvejes. F.eks.:

  - Specifikation af krav som f.eks. forretningskrav, funktionelle krav, systemkrav, userstories, epics, usecases eller lignende arbejdsprodukter, der beskriver de ønskede funktionelle eller ikke-funktionelle komponenter eller systemadfærd
  - Oplysninger om design og implementering som f.eks. diagrammer eller dokumenter, der beskriver system- eller softwarearkitektur, designspecifikationer, kaldegrafer, modeldiagrammer (f.eks. UML eller elementrelationsdiagrammer), grænsefladespecifikationer eller lignende arbejdsprodukter, der specificerer komponent eller systemstruktur
  - Implementeringen af komponenten eller systemet selv, herunder kode, database metadata og forespørgsler og grænseflader
  - Risikoanalyserapporter, der kan vurdere funktionelle, ikke-funktionelle og strukturelle aspekter af komponenten eller systemet
- Evaluering af testgrundlaget og testelementer for at identificere forskellige typer defekter som f.eks.:
  - Tvetydigheder
  - Udeladelser
  - Inkonsistenser
  - Unøjagtigheder
  - Selvmodsigelser
  - Overflødige udsagn
- Identificerbare egenskaber og sæt af egenskaber, der skal testes
- Definere og prioritere testbetingelser for hver egenskab på grundlag af analyser af testgrundlaget, og vurdere funktionelle, ikke-funktionelle og strukturelle karakteristikker, andre forretningsmæssige og tekniske faktorer og risikoniveauer
- Dokumenterer tovejs sporbarhed mellem hvert element i testgrundlaget og de tilknyttede testbetingelser (se afsnit 1.4.3 og 0)

Anvendelse af black-box, white-box og erfaringsbaserede testteknikker, kan være brugbare i løbet af testanalysen (se kapitel 4) for at reducere risikoen for at komme til at udelade vigtige testbetingelser og for at definere mere præcise og nøjagtige testbetingelser.

I nogle tilfælde producerer testanalysen testbetingelser, der skal anvendes som testens målsætninger i testcharters. Testcharters er typiske arbejdsprodukter i nogle typer erfaringsbaseret test (se sektion 4.4.2). Hvis disse målsætninger for testen er sporbare til testgrundlaget, kan den opnåede dækning for den slags erfaringsbaseret test måles.

Identifikation af defekter under testanalysen er en vigtig potentiel fordel, især hvis man ikke anvender nogen anden reviewproces og/eller testprocessen er tæt forbundet med reviewprocessen. Den slags testanalyseaktiviteter bekræfter, om kravene er konsistente, tilstrækkeligt udtrykte og færdiggjorte. De bekræfter også, om kravene imødekommer kunders, brugeres og interessenters behov på passende vis. Eksempelvis teknikker som Behavior Driven Development (BDD) og Accept Test Driven Development (ATDD). Disse indebærer, at der skal dannes testbetingelser og testcases ud fra userstories og acceptkriterier inden kodning. Disse teknikker finder, verificerer og validerer også defekter i userstories og accept-kriterier (se ISTQB® Foundation Level Agile Tester Extension syllabus).

Testdesign
I løbet af testdesignet udvides testbetingelserne til testcases på højt niveau, sæt af testcases på højt niveau og anden testware. Så testanalysen besvarer spørgsmålet "hvad der skal testes?", mens testdesignet besvarer spørgsmålet "hvordan det skal testes?".

Testdesignet indebærer følgende større aktiviteter:

- Design og prioritering af testcases og sæt af testcases
- Identifikation af nødvendige testdata for at understøtte testbetingelser og testcases

- Design af testmiljøet og identifikation af nødvendig infrastruktur og værktøjer
- At dokumentere tovejs sporbarhed mellem testgrundlaget, testbetingelser, testcases (se afsnit 1.4.4)

Udvidelsen af testbetingelser til testcases og sæt af testcases under testdesign indebærer ofte anvendelse af testteknikker (se kapitel 4).

Ligesom med testanalysen, kan testdesignet også resultere i identifikation af lignende typer af defekter i testgrundlaget. Og ligesom ved testanalysen kan identifikation af defekter under testdesignet også være vigtig og gavnlig for resultatet.

Testimplementering
I løbet af testimplementeringen udarbejdes/eller færdiggøres det nødvendige testware, herunder at placere testcases i rækkefølge for testprocedurerne. Testdesignet besvarer derfor spørgsmålet, "hvordan man tester det?", hvorimod implementeringen besvarer spørgsmålet "er alt på plads til at kunne udføre testene?"

Implementering af testene indebærer følgende større aktiviteter:

- At udvikle og prioritere testprocedurer og potentielt set skabe automatiserede testscripts
- At lave testsuites fra testprocedurer og (om nogen) automatiserede testmanuskripter
- At arrangere testsuites inden for testafviklingsplanen på en måde, der resulterer i en effektiv afvikling af testen (se afsnit 5.2.4)
- At opbygge testmiljøet (potentielt inklusiv testharness, servicevirtualisering, simulatorer og andre elementer af infrastrukturen) og verificere, at alt det nødvendige er blevet sat korrekt op
- At forberede testdata og sikre, at det er loadet korrekt ind i testmiljøet
- At verificere og opdatere tovejs sporbarhed mellem testgrundlaget, testbetingelserne, testcases, testprocedurer og testsuiter (se afsnit 1.4.4)

Testdesign og opgaver i forbindelse med implementering af test kombineres ofte.

I udforskende test og andre typer af erfaringsbaserede tests kan testdesign og implementering forekomme og kan blive dokumenteret under afvikling. Udforskende test baseres på testcharters (produceret som en del af testanalysen), og udforskende tests udføres med det samme, som de bliver designet og implementeret (se afsnit 4.4.2).

Testafvikling
Under afvikling af testen køres der testsuites i overensstemmelse med testens afviklingsplan.

Afvikling af testen indebærer følgende større aktiviteter:

- Dokumentering af testelemtentet(s) ID og version eller testobjektet, testværktøj(erne) og testwaren
- Afvikling af test manuelt eller ved hjælp af værktøjer
- Sammenligning af de faktiske resultater med de forventede resultater
- Analyse af anomalier for at finde de sandsynlige årsager (f.eks. kan der både forekomme fejl pga. defekter i koden og på grund af falske positiver (se afsnit 1.2.3))
- Rapportering af defekter på grundlag af observerede afvigelser (se afsnit 5.6)
- Logning af resultatet for den udførte test (f.eks. bestået, fejlet, blokeret)
- Gentagne testaktiviteter enten som et resultat af de handlinger, der udføres for en afvigelse, eller som en del af en planlagt test (f.eks. afvikling af en korrigeret test, gentest og/eller regressionstest)
- Verifikation og opdatering af tovejs sporbarhed mellem testgrundlag, testbetingelser, testcases, testprocedurer og testens resultater

Testafslutning
Aktiviteter forbundet med testens færdiggørelse indsamler data fra de færdiggjorte testaktiviteter for at forene erfaringer med testware og andre relevante oplysninger. Disse aktiviteter finder sted ved milepæle for projektet, f.eks. når man leverer softwaresystemet til produktion, når man afslutter (eller aflyser) et testprojekt, når man leverer en agil projektiteration, når et testniveau er afsluttet, eller man har udarbejdet et vedligeholdelsesdokument.

Testens afslutning indebærer følgende større aktiviteter:

- Tjekke, at alle defektrapporter er lukket, tilføje ændringsønsker eller product backlog items for defekter, der ikke er løst ved testens afslutning
- Udarbejdelse af en testopsummeringsrapport, der skal kommunikeres til interessenter
- Færdiggørelse og arkivering af testmiljø, testdata, testens infrastruktur og anden testware til senere anvendelse
- Overdragelse af testware til vedligeholdelsesteams, til andre projektteams og/eller til andre interessenter, der kan have gavn af det
- Analyse af de erfaringer, man har gjort fra de færdiggjorte testaktiviteter for at afgøre, om der er behov for ændringer i fremtidige iterationer, leverancer og projekter
- Anvendelse af de indsamlede oplysninger for at forbedre testprocessens modenhed

1.4.3 Testens arbejdsprodukter

Testens arbejdsprodukter er udarbejdet som en del af testprocessen. Ligesom der er betydelig variation i den måde, hvorpå organisationer implementerer testprocessen, er der også betydelig variation i den type arbejdsprodukter, som bliver skabt i løbet af processen, måden, som arbejdsprodukterne bliver organiseret og styret på, og de navne, der anvendes til dem. Denne syllabus følger de testprocesser, som er blevet skitseret ovenfor, og de arbejdsprodukter, der er beskrevet i denne syllabus og i ISTQB®'s ordliste. ISO-standarden (ISO/IEC/ IEEE 29119-3) kan også fungere som en rettesnor for testens arbejdsprodukter.

Mange af testens arbejdsprodukter, der er beskrevet i dette afsnit, kan lagres og styres vha. teststyringsværktøjer og defektstyringsværktøjer (se kapitel 6).

Testplanlægningens arbejdsprodukter
Testplanlægningens arbejdsprodukter omfatter typisk en eller flere testplaner. Testplanen indeholder oplysninger om det testgrundlag, som de andre af testens arbejdsprodukter er tilknyttet via sporbarhedsoplysninger (se nedenfor og afsnit 1.4.4), såvel som slutkriterier (eller Definition of Done), som kan anvendes i løbet af testovervågning og -kontrol. Testplanerne beskrives i afsnit 5.2.

Arbejdsprodukter fra testovervågning og -kontrol
Arbejdsprodukterne til testovervågning og -kontrol omfatter typisk mange forskellige testrapporter, herunder testfremdriftsrapporter (der udfærdiges på løbende og/eller regulær basis) og testopsummeringsrapporter (der udfærdiges ved forskellige milepæle). Alle testrapporter bør indeholde interessentspecifikke detaljer om testens fremdrift fra rapportens data, inklusive resume af resultaterne af den udførte test, når de er tilgængelige.

Både testovervågning og kontrolarbejdsprodukter bør også tage hensyn til projektledelsens overvejelser og indsats f.eks. at færdiggøre opgaven, at fordele resurser og anvendelsen.

Testovervågning og kontrol, og de arbejdsprodukter, der bliver til via disse aktiviteter, bliver yderligere beskrevet i afsnit 5.3 af denne syllabus.

Testanalysens arbejdsprodukter
Testanalysens arbejdsprodukter indebærer specificerede og prioriterede testbetingelser, hvor de ideelt set hver især er tovejs sporbare til de(t) specifikke element(er) i det testgrundlag, den dækker. I udforskende test

kan testanalyse omfatte udarbejdelse af testcharters. Testanalysen kan også resultere i opdagelsen og rapporteringen af defekter i testgrundlaget.

Testdesignets arbejdsprodukter
Testdesignet leder til testcases og sæt af testcases, der kan efterprøve de testbetingelser, som er defineret i testanalysen. Det anses ofte som god praksis at designe testcases på højt niveau uden konkrete værdier til indtastede data og forventede resultater. Sådanne testcases kan genbruges gennem adskillige testcyklusser med forskellige typer konkrete data, mens man samtidig dokumenterer testcasens omfang. Ideelt set er hver testcase tovejs sporbar til den/de testbetingelse(r), den dækker.

Testdesignet leder også til:

- Design og/eller identifikation af de nødvendige testdata
- Design af testmiljø
- Identifikation af infrastruktur og værktøjer

Selvom omfanget af resultaterne kan variere betydeligt.

Testimplementeringens arbejdsprodukter
Testimplementeringens arbejdsprodukter omfatter:

- Testprocedurer og sekvenser af disse testprocedurer
- Testsuiter
- En tidsplan for testafviklingen

Ideelt set, når implementering af testen er udført, kan man demonstrere, at dækningskriteriet i testplanen er opnået via tovejs sporbarhed mellem testprocedurerne og de specifikke elementer i testgrundlaget, gennem testcases og testbetingelser.

I nogle tilfælde indebærer testimplementeringen udfærdigelsen af arbejdsprodukter ved at anvende eller til anvendelse af værktøjer, som f.eks. servicevirtualisering og automatiserede testscripts.

Testimplementering kan også lede til udfærdigelse og verifikation af testdata og testmiljø. Kvaliteten af de dokumenterede resultater for data og/eller miljø kan variere betydeligt.

Testdataene anvendes til at tildele konkrete værdier til de pågældende testcases' input og forventede resultater. Den slags konkrete værdier, samt eksplicitte angivelser om anvendelsen af konkrete værdier, gør testcases på højt niveau til afviklingsbare testcases på detaljeret niveau. Samme testcase på højt niveau kan anvende andre testdata, når den afvikles på forskellige udgaver af testobjektet. Det konkrete forventede resultat, der er tilknyttet de konkrete testdata, identificeres ved brug af et testorakel.

I en udforskende test vil noget testdesign og implementering af arbejdsprodukter blive udarbejdet under testens afvikling, men i hvilket omfang den udforskende test (og dens sporbarhed til specifikke elementer i testgrundlaget) dokumenteres kan variere betydeligt.

Testbetingelserne, der står defineret i testanalysen kan tilpasses yderligere i implementering af testen.

Testafviklingens arbejdsprodukter
Testafviklingens arbejdsprodukter omfatter:

- Dokumentation af status på individuelle testcases eller testprocedurer (f.eks. klar til afvikling, bestået, fejlet, blokeret, bevidst sprunget over osv.)

- Defektrapporter (se afsnit 5.6)
- Dokumentation af den eller de testelementer, testobjekter, testværktøjer og testware, der indgik i testen

Når testen er udført, kan status på hvert af testgrundlagets elementer ideelt set afgøres og rapporteres via tovejs sporbarhed med den eller de tilknyttede testprocedurer. F.eks. kan vi afgøre, hvilke krav der er godkendt i alle planlagte tests, hvilke krav der er fejlet i testene og/eller har tilknyttede defekter, og hvilke krav, der stadig afventer afvikling af tests. Dette muliggør verifikation af, om dækningskriteriet er blevet mødt, og det muliggør rapportering af testresultater i termer, der er forståelige for interessenterne.

Testafslutningens arbejdsprodukter
Arbejdsprodukterne ved testens afslutning inkluderer testopsummeringsrapporter, handlingspunkter til efterfølgende projekter eller iterationer, ændringsønsker eller product backlog-items og færdiggjort testware.

1.4.4 Sporbarheden mellem testgrundlag og testens arbejdsprodukter

Som nævnt i afsnit 1.4.3, kan testens arbejdsprodukter og navnene på arbejdsprodukterne variere betydeligt. Til trods for disse variationer er det vigtigt at etablere og opretholde sporbarhed igennem testprocessen for hvert af testgrundlagets elementer og testens forskellige arbejdsprodukter. Arbejdsprodukterne er forbundet med elementet som beskrevet ovenfor, for på effektiv vis at kunne implementere overvågning og kontrol. Ud over evaluering af testdækningen, understøtter god sporbarhed desuden:

- Analyse af effekten af ændringernes ændringer
- At gøre testen reviderbar
- At imødekomme krav til IT-forvaltning
- Forbedring af testfremdriftsrapporternes og testopsummeringsrapporternes forståelighed. så status på elementerne i testgrundlaget inkluderes (f.eks. krav, der har bestået deres tests; krav, der er fejlet i test; og krav, der afventer test)
- At formidle de tekniske aspekter af test til interessenter, så de kan forstå det
- At levere oplysninger til vurdering af produktkvalitet, procesevne og projektets fremdrift imod forretningens mål

Nogle teststyringsværktøjer leverer arbejdsproduktskabeloner til testen, der passer til nogle eller alle dele af testens arbejdsprodukter, som er skitseret i dette afsnit. Nogle organisationer bygger deres egne styringssystemer til at organisere arbejdsprodukterne og den nødvendige informationssporbarhed.

1.5 Testens psykologi

Softwareudvikling, herunder softwaretests, involverer mennesker. Derfor påvirker den menneskelige psyke også softwaretests.

1.5.1 Test og den menneskelige psyke

Hvis man identificerer defekter i forbindelse med den statiske test, som f.eks. ved at granske kravspecifikationer, ved forfining af userstories, eller hvis man finder afvigelser under afvikling af den dynamiske test, kan det opfattes som kritik af produktet og den, der har fremstillet det. Der er et element i menneskets psykologi, som kaldes for bekræftelsesbias. Det henviser til, at det kan være svært at acceptere oplysninger, som modsiger ens nuværende overbevisninger. Hvis f.eks. udviklere forventer, at deres kode fungerer korrekt, kan de være præget af denne bekræftelsesbias, der gør det svært for dem at acceptere, at koden ikke er korrekt. Ud over bekræftelsesbias kan andre kognitive fordomme gøre det svært for folk at forstå eller acceptere de oplysninger, der fremkommer i forbindelse med test. Ydermere er det et typisk menneskeligt træk, at vi bebrejder den, der leverer de dårlige nyheder. De oplysninger som opstår ved test, indeholder ofte dårligt nyt.

Som resultat af disse psykologiske faktorer anser nogle test for at være en destruktiv aktivitet, selvom testen i høj grad bidrager til projektets fremdrift og produktets kvalitet (se afsnittene 1.1 og 1.2). For at afhjælpe den opfattelse bør man kommunikere oplysninger om defekter og afvigelser på en konstruktiv måde. Således kan man reducere spændinger mellem testere, analytikere, product owners, designere og udviklere. Dette gælder både ved statisk og dynamisk test.

Testere og testmanagers skal have gode mellemmenneskelige færdigheder for at kunne kommunikere effektivt om defekter, afvigelse, testresultater, testens fremdrift og risici, og for at kunne opbygge et positivt forhold til kollegerne. Blandt gode måder at kommunikere på tæller følgende eksempler:

- Begynd med samarbejde, i stedet for konflikt. Mind alle om, at det fælles mål er at forbedre systemets kvalitet.
- Læg vægt på fordele ved testen. F.eks. kan defektoplysninger hjælpe forfatterne med at forbedre deres arbejdsprodukter og færdigheder. For organisationen vil det spare tid og penge og reducere de overordnede risici ved produktkvaliteten, hvis defekter findes og udbedres.
- Viderebring testresultater og andre fund på en neutral og faktaorienteret måde uden at kritisere personen, der har udarbejdet den defekte genstand. Skriv objektive og faktuelle defektrapporter og gennemgå fundene.
- Sæt dig ind i, hvad den anden person føler og årsagerne til, at denne kan reagere negativt på informationerne
- Få bekræftet, at den anden person har forstået det, der er blevet sagt, og omvendt

Typiske målsætninger med testen er blevet beskrevet ovenfor (se afsnit 1.1). At definere det rette sæt målsætninger for testen rummer vigtige psykologiske følger. De fleste vil forsøge at forene deres planer og adfærd med teamet, ledelsen og interessenternes målsætninger. Det er også vigtigt, at testerne følger disse målsætninger med færrest mulige personlige fordomme.

1.5.2 Testerens og udviklerens mindset

Udviklere og testere tænker ofte forskelligt. Udviklingens primære formål er at designe og bygge et produkt. Som det blev forklaret tidligere, omfatter test verifikation og validering af produktet, at finde defekter inden leverancen osv. Det er forskellige målsætninger, der kræver forskellige mentaliteter. Hvis disse mentaliteter forenes, kan man opnå produktkvalitet med højt niveau.

Mentaliteten afspejler individets antagelser og foretrukne metoder, når denne tager beslutninger og løser problemer. En testers mentalitet bør involvere nysgerrighed, professionel pessimisme, en kritisk sans, opmærksomhed på detaljerne og motivation til god og positiv kommunikation og et godt forhold til andre. En testers mentalitet vil typisk udvikle sig og modnes med testerens erfaring.

En udviklers mentalitet kan omfatte nogle af de samme elementer som testerens, men udviklere er ofte mere interesseret i at designe og bygge løsninger, end i at overveje, hvad der kunne være galt med disse løsninger. Derudover kan bekræftelsesbiaset gøre det vanskeligt at være opmærksom på ens egne fejl.

Med den rigtige mentalitet vil en udvikler være i stand til at teste sin egen kode. Forskellige softwareudviklingslivscyklusser indebærer ofte forskellige måder at organisere testere og testaktiviteter på. Hvis nogle af testaktiviteterne udføres af uafhængige testere, vil det øge sandsynligheden for fund af defekter. Det er særligt vigtigt for store, komplekse eller sikkerhedskritiske systemer. Uafhængige testere bringer et perspektiv med sig, der er anderledes end arbejdsproduktforfatterens eget perspektiv (f.eks. forretningsanalytikere, designere og udviklere), da de har anderledes kognitive fordomme end forfatterne.

2. Test igennem hele softwareudviklings-livscyklussen — 100 minutter

Nøgleord

accepttest, alfatest, betatest, brugeraccepttest, driftmæssig accepttest, effektanalyse, funktionelle test, gentest, ikke-funktionelle test, integrationstest, kommerciel standardsoftware (COTS), komponentintegrationstest, komponenttest, kontraktuel accepttest, regressionstest, regulativ accepttest, sekventiel udviklingsmodel, systemintegrationstest, systemtest, testcase, testformål, testgrundlag, testmiljø, testniveau, testobjekt, testtype, vedligeholdelsestest, white-box test, ændringsrelaterettest

Læringsmål for tests igennem hele softwareudviklingslivscyklussen

2.1 Softwareudviklingens livscyklusmodeller

FL-2.1.1 (K2) Forklar relationen mellem softwareudviklingens aktiviteter og testaktiviteter i softwareudviklingens livscyklus

FL-2.1.2 (K1) Beskriv årsagerne til, at softwareudviklingens livscyklusmodeller skal tilpasses til projektets kontekst og produktets karakteristika

2.2 Testniveauer

FL-2.2.1 (K2) Sammenlign de forskellige testniveauer ud fra målsætningen, testgrundlaget, de typiske defekter og afvigelser, tilgange og ansvarsområder

2.3 Testtyper

FL-2.3.1 (K2) Sammenlign funktionelle, ikke-funktionelle og white-box test

FL-2.3.2 (K1) Anerkend at funktionelle, ikke-funktionelle og white-box test, kan finde sted på ethvert testniveau

FL-2.3.3 (K2) Sammenlign formålet med gentest og regressionstest

2.4 Vedligeholdelsestest

FL-2.4.1 (K2) Opsummer de udløsende faktorer for vedligeholdelsestest

FL-2.4.2 (K2) Beskriv effektanalysens rolle i vedligeholdelsestest

2.1 Softwareudviklingslivscyklusmodeller

En softwareudviklingslivscyklusmodel beskriver de typer af aktiviteter, der udføres i hvert stadie af et softwareudviklingsprojekt, og hvordan aktiviteterne forholder sig til hinanden logisk og kronologisk. Der er et antal af forskellige softwareudviklingslivscyklusmodeller, der hver især kræver forskellige tilgange til test.

2.1.1 Softwareudvikling og softwaretest

Det er en vigtig del af testerens rolle at være bekendt med de almindelige softwareudviklingslivscyklus-modeller, så de passende testaktiviteter kan finde sted.

Inden for enhver af softwareudviklingens livscyklusmodeller, findes der flere karakteristika, der beskriver gode tests:

- For enhver udviklingsaktivitet, findes der er en tilsvarende testaktivitet
- For hvert testniveau findes der testmålsætninger til det pågældende niveau
- Testanalyse og -design til et givet testniveau begynder i løbet af den tilsvarende udviklingsaktivitet
- Testere deltager i diskussioner for at definere og forfine krav og design, og de deltager i review af arbejdsprodukterne (f.eks. krav, design, userstories osv.) lige så snart udkast er tilgængelige

Uanset hvilken livscyklusmodel for softwareudviklingen, der vælges, bør testaktiviteterne begynde i de tidligere stadier af livscyklussen for at følge testprincippet om tidlig test.

Denne syllabus opdeler de almene softwareudviklingslivscyklusmodeler således:

- Sekventielle udviklingsmodeller
- Inkrementelle og iterative udviklingsmodeller

En sekventiel udviklingsmodel beskriver softwareudviklingen som et lineært og sekventiel aktivitetsflow. Det betyder, at enhver fase i udviklingsprocessen bør begynde, når den forrige fase er færdiggjort. I teorien er der ingen overlap i faserne, men i praksis er det gavnligt at modtage tidlig feedback fra den følgende fase.

I vandfaldsmodellen færdiggøres udviklingsaktiviteterne (f.eks. kravanalyse, design, kodning, tests) en efter en. I denne model finder testaktiviteterne kun sted efter alle andre udviklingsaktiviteter er færdiggjorte.

I modsætning til vandfaldsmodellen indarbejder V-modellen testprocessen igennem udviklingsprocessen ved at implementere princippet om tidlig test. Derudover inkluderer V-modellen de testniveauer, der er relateret til hver eneste af de tilsvarende udviklingsfaser, hvad der yderligere understøtter tidlig test (se afsnit 2.2 for en diskussion af testniveauer). I denne model udføres testene, der er relateret til hvert testniveau sekventielt, men i nogle tilfælde kan der forekomme overlap.

Sekventielle udviklingsmodeller leverer software, der indeholder komplette sæt af funktioner, men som typisk kræver måneder eller år før levering til interessenter og brugere.

Inkrementel udvikling indebærer at fastslå krav, design, opbygning og test af systemet i enkeltdele, hvilket betyder, at softwarens funktioner udvikles inkrementelt. Størrelsen på disse funktioners inkrementer varierer, hvor der i nogle metoder anvendes større dele og i andre mindre dele. Funktionsinkrementerne kan være helt ned til en enkelt ændring af brugergrænsefladens skærmbillede eller en ny spørgemulighed.

Iterativ udvikling finder sted, når en gruppe funktioner specificeres, designes, opbygges og testes sammen i en serie af cyklusser, ofte inden for en fastsat periode. Iterationer kan indeholde forandringer af funktioner, der er udviklet i tidligere iterationer, sammen med forandringer inden for projektets omfang. Hver eneste iteration

leverer fungerende software, der er en voksende undergruppe af de samlede funktioner, inden den endelige software leveres, eller udviklingen stoppes.

Eksempler på dette omfatter:

- Rational Unified Process: Hver iteration er ofte relativt lang (f.eks. to til tre måneder) og indeholder dele, der er tilsvarende store, så som to eller tre grupper af relaterede funktioner
- Scrum: Hver iteration er ofte relativt kort (f.eks. timer, dage eller et par uger) og indeholder funktionsinkrementer, der er tilsvarende små, så som få forbedringer og/eller to eller tre nye funktioner
- Kanban: Implementeret med eller uden iterationer med fastslået længde, der kan levere enten en enkelt forbedring eller funktion uden færdiggørelse eller kan gruppere funktioner sammen til samtidig udgivelse
- Spiral: Indebærer udarbejdning af eksperimentelle inkrementer, hvor nogle af disse kan undergå store ændringer eller endda blive opgivet i følgende udviklingsarbejde

Komponenter eller systemer, der anvender disse metoder, anvender ofte overlappende og gentagne testniveauer igennem udviklingen. Ideelt set bliver hver funktion testet på adskillige testniveauer, mens den nærmer sig levering. I nogle tilfælde kan teams anvende continuous delivery eller continuous integration, hvoraf begge indebærer betydelig automatisering på adskillige testniveauer, som en del af deres leverancepipelines. I mange softwareudviklingsorganisationer, hvor disse metoder anvendes, anvendes ligeledes konceptet om selvorganiserende teams. Dette kan ændre den måde testarbejdet organiseres på, såvel som forholdet mellem testere og udviklere.

Disse metoder danner et voksende system, hvor funktionerne kan leveres til slutbrugere enkeltvis, per iteration eller i en mere traditionel facon som en større leverance. Uanset om softwareinkrementerne leveres til slutbrugere, bliver regressionstest vigtigere, efterhånden som systemet vokser.

I modsætning til de sekventielle modeller kan iterative og inkrementelle modeller levere anvendelig software på uger eller endda dage, men kan kun levere det komplette sæt af påkrævede produkter over en periode på måneder eller endda år.

For flere oplysninger om softwaretests i konteksten af Agile-udvikling, se ISTQB® CTFL-AT Certified Tester Foundation Level Agile Tester syllabus, Black 2017, Crispin 2008 og Gregory 2015.

2.1.2 Softwareudviklingslivscyklusmodeller i kontekst

Softwareudviklingslivscyklussens modeller skal vælges og tilpasses til projektets kontekst og produktets karakteristika. En passende softwareudviklingslivscyklusmodel bør vælges og tilpasses ud fra projektets mål, hvilken type produkt, der udvikles, forretningsprioriteter (f.eks. produktionstid) og identificerede produkt- og projektrisici. F.eks. bør udviklingen og tests af mindre og interne administrative systemer adskille sig fra udviklingen og test af sikkerhedskritiske systemer, så som en bils bremsekontrolsystem. Som et andet eksempel kan organisatoriske og kulturelle udfordringer i nogle tilfælde forhindre kommunikationen mellem teammedlemmer, hvilket kan hæmme den iterative udvikling.

Afhængigt af projektens kontekst, kan det være nødvendigt at kombinere eller omorganisere testniveauer og/eller testaktiviteter. F.eks. ved integrationen af et kommercielt standardsoftware (COTS) ind i et større system, kan køberen udføre interoperationalitetstest på systemintegrationsniveau (f.eks. integration ind i infrastruktur og andre systemer) og på accepttestniveau (funktionelt og ikke-funktionelt sammen med brugeracceptest og driftsaccepttest). Se afsnit 2.2 for en gennemgang af testniveauer og afsnit 2.3 for en gennemgang af testtyper.

Derudover kan softwareudviklingens livscyklusmodeller selv kombineres. F.eks. kan en V-model anvendes til udvikling og test af backend-systemer og deres integrationer, mens en agil udviklingsmodel kan anvendes til

at udvikle og teste front-end brugerfladen (UI) og funktionaliteten. Prototyping kan anvendes tidligt i et projekt ved at indføre en inkrementel udviklingsmodel, når den eksperimentelle fase er overstået.

Internet of Things (IoT)-baserede systemer, der består af mange forskellige objekter, f.eks. enheder, produkter og tjenester, benytter typisk adskilte livscyklusmodeller i softwareudviklingen for hvert objekt. Dette leder til særlige udfordringer i udviklingen af systemversioner for Internet of Things. Derudover lægger den slags objekters softwareudviklingslivscyklus større vægt på de senere faser i softwareudviklingslivscyklussen, efter de er blevet sat i drift (f.eks. drift, opdatering og nedlukning).

Softwareudviklingsmodeller skal tilpasses konteksten i projektet og produktets karakteristika af flere grunde:

- Forskelle i produktrisiko for systemet (komplekse eller simple projekter)
- Mange forretningsenheder kan være del af et projekt eller program (kombination af sekventiel og agil udvikling)
- Kort tid til at leverer et produkt til markedet (sammenlægning af testniveauer og/eller integration af testtyper i testniveauer)

2.2 Testniveauer

Testniveauer er grupperinger af testaktiviteter, som er organiseret, og som styres samlet. Hvert testniveau er en forekomst i testprocessen, der består af de aktiviteter, som står beskrevet i afsnit 1.4 og bliver udført i overensstemmelse med software på et givet niveau i udviklingen ud fra individuelle enheder eller komponenter for at fuldende systemer eller, hvor det gør sig gældende, systemer af systemer. Testniveauer relaterer til andre aktiviteter inden for softwareudviklingens livscyklus. De testniveauer, der anvendes i denne syllabus, er:

- Komponenttest
- Integrationstest
- Systemtest
- Accepttest

Testniveauer karakteriseres af de følgende egenskaber:

- Specifikke formål
- Testgrundlag, henvises til for at udlede testcases
- Testobjekt (dvs. det, der testes)
- Typiske defekter og afvigelser
- Særlige tilgange og ansvar

For alle testniveauer kræves der et passende testmiljø. F.eks. i accepttest er et produktionslignende testmiljø ideelt, hvorimod udviklerne typisk anvender eget udviklingsmiljø til komponenttest.

2.2.1 Komponenttest

Komponenttestens formål

Komponenttest (også kendt som unit- eller modultests) fokuserer på komponenter, der kan testes separat. Komponenttestens formål omfatter:

- Reduktion af risiko
- Bedømmelse af om en komponents funktionelle og ikke-funktionelle adfærd er designet og specificeret som ønsket
- Opbygge tillid til komponentens kvalitet
- Finde defekter i komponenten

- Forhindre defekter fra at slippe igennem til højere testniveauer

I nogle tilfælde, særligt i inkrementelle og iterative modeller (f.eks. Agile), hvor kodeændringer løbende finder sted, spiller automatiserede komponentregressionstests en central rolle i opbygning af tillid til, at forandringerne ikke har ødelagt de eksisterende komponenter.

Komponenttests udføres ofte isoleret fra resten af systemet afhængigt af softwareudviklingens livscyklusmodel og systemet, hvor det kan være nødvendigt at anvende simulerede objekter, servicevirtualisering, harness, stubbe og drivere. Komponenttests kan dække funktionalitet (f.eks. beregningernes korrekthed), ikke-funktionelle karakteristika (f.eks. søgning efter memory leaks) og strukturelle egenskaber (f.eks. beslutnings-test).

Testgrundlag
Eksempler på arbejdsprodukter, der kan anvendes som testgrundlag for komponenttests omfatter:

- Detaljeret design
- Kode
- Datamodel
- Komponentspecifikation

Testobjekter
Typiske testobjekter i komponenttest omfatter:

- Komponenter, units eller moduler
- Kode eller datastrukturer
- Klasser
- Databasemoduler

Typiske defekter og afvigelser
Eksempler på typiske defekter og afvigelser i komponenttests omfatter:

- Ukorrekt funktionalitet (f.eks. ikke som beskrevet i designspecifikation)
- Problemer med dataflow
- Ukorrekt kode og logik

Defekter bliver typisk udbedret, så snart de findes, og ofte uden en formel defekthåndtering. Men når udviklere rapporterer defekter, kan det bidrage med oplysninger til at finde rodårsager og forbedre processen.

Særlige tilgange og ansvarsområder
Komponenttests udføres normalt af den udvikler, der har skrevet koden. Komponenttests kræver som minimum adgang til den kode, som testes. Udviklere kan finde afveksling i komponentudviklingen ved i stedet at lede efter defekter og udbedre dem. Udviklere vil ofte skrive og afvikle test efter at have skrevet koden til en komponent. Særligt i agil udvikling kan skrivning af automatiserede komponenttestcases gå forud for skrivning af applikations-koden.

F.eks. i tilfælde af test driven development (TDD). Testdrevet udvikling er i høj grad iterativt og er baseret på udvikling af automatiserede testcases i cyklusser, derefter opbygge og integrere små dele af koden, afvikle komponenttests, rette problemerne og omstrukturere koden. Denne proces fortsætter, indtil komponenten er blevet færdigbygget, og alle komponenttests er bestået. Testdrevet udvikling er et eksempel på en test-først tilgang. Selvom testdrevet udvikling opstod i eXtreme Programming (XP), har det spredt sig til andre former for Agile og også til sekventielle livscyklusser (se ISTQB® CTFL-AT Certified Tester Foundation Level Agile Tester syllabus).

2.2.2 Integrationstest

Integrationstestens formål
Integrationstests fokuserer på interaktionen mellem komponenter eller systemer. Formål for integrationstest omfatter:

- Reduktion af risiko
- Verifikation af om grænsefladens funktionelle og ikke-funktionelle adfærd er designet og specificeret som ønsket
- Opbygning af tillid til grænsefladernes kvalitet
- Finde defekter (hvilket kan forekomme i selve grænsefladerne eller inden for komponenterne eller systemerne)
- Forhindre defekter fra at slippe igennem til højere testniveauer

Ligesom ved komponenttest kan automatiseret integrationstest skabe tillid til, at ændringerne ikke har ødelagt de eksisterende grænseflader, komponenter eller systemer.

I denne syllabus er beskrevet to forskellige niveauer af integrationstest, som kan udføres på testobjekter af forskellig størrelse:

- Integrationstests af komponenter fokuserer på interaktioner og grænseflader mellem integrerede komponenter. Integrationstest af komponenter udføres efter komponenttesten og er generelt automatiseret. I iterativ og inkrementel udvikling er integrationstest af komponenter normalt en del af den kontinuerlige integrationsproces
- Systemintegrationstests fokuserer på interaktioner og grænseflader mellem systemer, pakker og microservices. Systemintegrationstests kan også dække integrationer med og grænseflader leveret af eksterne organisationer (f.eks. internettjenester). I dette tilfælde kontrollerer den udviklende organisation ikke de eksterne grænseflader, som kan skabe forskellige udfordringer i tests (f.eks. sikre at testblokerende defekter i den eksterne organisations kode er udbedret, planlægning af test-miljø osv.). Systemintegrationstest kan færdiggøres efter systemtest eller parallelt med i gang-værende systemtestaktiviteter (både i sekventiel udvikling og i iterativ og inkrementel udvikling)

Testgrundlag
Eksempler på arbejdsprodukter, der kan anvendes som testgrundlag for integrationstest omfatter:

- Software og systemdesign
- Sekvensdiagrammer
- Grænsefladens og kommunikationsprotokollens specifikationer
- Usecases
- Arkitektur på komponent- eller systemniveau
- Arbejdsflow
- Eksterne definitioner for grænsefladen

Testobjekter
Typiske testobjekter i integrationstest omfatter:

- Delsystemer
- Databaser
- Infrastruktur
- Grænseflader
- API'er
- Microservices

Typiske defekter og afvigelser
Eksempler på typiske defekter og afvigelser i integrationstest for komponenter omfatter:

- Ukorrekte data, manglende data eller ukorrekt datakodning
- Ukorrekt sekventering eller timing af grænsefladekald
- Grænseflader, der ikke stemmer overens
- Afvigelser i kommunikationen mellem komponenter
- Ikke-håndterede eller forkert håndterede afvigelser i kommunikation mellem komponenter
- Ukorrekte formodninger om betydning, units eller grænser for data, der overføres mellem komponenter

Eksempler på typiske defekter og afvigelser for systemintegrationstests omfatter:

- Inkonsistente beskedstrukturer mellem systemer
- Ukorrekt data, manglende data eller ukorrekt datakodning
- Uoverensstemmende grænseflader
- Afvigelser i kommunikationen mellem systemer
- Afvigelser på grundlag af ikke-håndteret eller forkert håndteret kommunikation mellem systemer
- Fejlagtige formodninger om betydning, enheder eller grænser for data, der overføres mellem systemer
- Manglende overholdelse af obligatoriske sikkerhedsregler

Særlige tilgange og ansvarsområder
Komponentintegrationstest og systemintegrationstest bør have fokus på selve integrationen. F.eks. hvis modul A integreres med modul B, bør testen fokusere på kommunikationen mellem modulerne, ikke på de individuelle modulers funktionalitet, da dette bør dækkes i løbet af komponenttest. Hvis system X integreres med system Y, bør testene fokusere på kommunikationen mellem systemerne, ikke på de individuelle systemers funktionalitet, da dette bør dækkes i løbet af systemtest. Funktionelle, ikke-funktionelle og strukturelle testtyper er anvendelige.

Komponentintegrationstest er ofte udviklernes ansvar. Systemintegrationstest er generelt testernes ansvar. Ideelt set bør de testere, der udfører systemintegrationstesten forstå systemarkitekturen og bør have indflydelse på planlægning af integrationen.

Hvis integrationstestene og integrationsstrategien planlægges før komponenterne eller systemerne er bygget, kan disse komponenter eller systemer blive bygget i den rækkefølge, der er påkrævet for at opnå den mest effektive test. Systematiske integrationsstrategier kan være baseret på systemarkitekturen (f.eks. top-down og bottom-op), funktionelle opgaver, transaktionsbehandlingens sekvenser eller andre aspekter af systemet eller komponenterne. For at kunne simplificere defekt isolering og spore defekter tidligt, bør integrationen normalt foregå inkrementelt (dvs. med et lavt antal af yderligere komponenter eller systemer ad gangen), snarere end "big bang" (dvs. integration af alle komponenter eller systemer i et enkelt trin). En risikoanalyse af de mest komplekse brugergrænseflader kan hjælpe med at indsnævre integrationstesten.

Jo bredere omfanget af integrationen er, desto sværere bliver det at isolere defekter til en specifik komponent eller et specifikt system. Det kan føre til at fejlfindingen bliver ramt af øgede risici og tager længere tid. Det er en årsag til, at continuous integration, hvor softwarens komponenter integreres enkeltvis (dvs. funktionel integration) er blevet almen praksis. Sådan continuous integration indebærer ofte automatiske regressionstests, ideelt set på flere testniveauer.

2.2.3 Systemtest

Systemtestens formål
Systemtest fokuserer på hele systemet, dets adfærd og kapacitet og tager ofte højde for de end-to-end opgaver, systemet kan udføre, og den ikke-funktionelle adfærd, det viser, mens det udfører opgaverne. Systemtestens formål omfatter:

- Reduktion af risiko
- Verifikation af om systemets funktionelle og ikke-funktionelle adfærd følger design og specifikationer
- Validering af om systemet er komplet og lever op til forventningerne
- Opbygning af tillid til systemets kvalitet som helhed
- Sporing af defekter
- Forhindre defekter i at slippe videre til højere testniveauer eller produktion

For visse systemer kan verifikation af datakvaliteten også være en målsætning. Ligesom med komponenttest og integrationstest, kan automatiserede systemregressionstest i nogle tilfælde øge tilliden til, at forandringerne ikke har ødelagt eksisterende egenskaber eller end-to-end funktionaliteter. Systemtest producerer ofte oplysninger, der anvendes af interessenter til at foretage beslutninger om release. Systemtest kan også leve op til love, regelkrav eller industristandarder.

Testmiljøet bør ideelt set stemme overens med slutmålet eller produktionsmiljøet.

Testgrundlag
Eksempler på arbejdsprodukter, der kan anvendes som testgrundlag for systemtest omfatter:

- System- og softwarekravenes specifikationer (funktionelle og ikke-funktionelle)
- Risikoanalyserapporter
- Usecases
- Epics og userstories
- Modeller af systemadfærd
- Tilstandsdiagrammer
- System- og brugervejledninger

Testobjekter
Typiske testobjekter i systemtest omfatter:

- Applikationer
- Hardware-/softwaresystemer
- Operativsystemer
- System under test (SUT)
- Systemkonfiguration og konfigurationsdata

Typiske defekter og afvigelser
Eksempler på typiske defekter og afvigelser for systemtest omfatter:

- Ukorrekte beregninger
- Systemets ukorrekte eller uventede funktionelle eller ikke-funktionelle adfærd
- Ukorrekt styring og/eller dataflows inden for systemet
- Systemet kan ikke udføre funktionelle opgaver end-to-end på passende vis
- Systemet fungerer ikke i systemmiljøet/-miljøerne
- Systemet fungerer ikke som beskrevet i system- og brugervejledninger

Særlige tilgange og ansvarsområder
Systemtest bør fokusere på systemets overordnede end-to-end adfærd som helhed, både funktionelt og ikke-funktionelt. Systemtest bør anvende de mest passende teknikker (se kapitel 4) på det aspekt af systemet, der skal testes. F.eks. kan der laves en beslutningstabel for at verificere, om den funktionelle adfærd passer til beskrivelsen i forretningsreglerne.

Systemtest udføres typisk af uafhængige testere der er meget afhængige af specifikationer. Defekter i specifikationerne (f.eks. manglende userstories, ukorrekt angivet forretningskrav osv.) kan lede til manglende forståelse af - eller uenighed om - den forventede systemadfærd. Den slags situationer kan forårsage falske positiver og falske negativer, der henholdsvis spilder tiden og reducerer effektiviteten i sporingen af defekter. Tidlig involvering af testere i refinement af userstories eller statiske testaktiviteter, så som reviews, hjælper med at begrænse, at den slags situationer opstår.

2.2.4 Accepttest

Accepttestens formål
Accepttest fokuserer ligesom systemtests typisk på hele systemet og dets adfærd og evner. Accepttestens formål omfatter:

- Opbygning af tillid til systemets kvalitet som helhed
- Validering af, om systemet er komplet og lever op til forventningerne
- Verifikation af, at systemets funktionelle og ikke-funktionelle adfærd følger specifikationerne

Accepttest kan give oplysninger til at vurdere, hvor tæt systemet er på at være klar til implementering og anvendelse af kunden (slutbruger). Det kan ske at man finder defekter ved accepttesten, men at finde defekter er ofte ikke en målsætning, og hvis et betydeligt antal defekter opdages i løbet af accepttesten, kan det i nogle tilfælde udgøre en større risiko for projektet. Accepttest kan også gennemføres for at opfylde lovkrav, regler eller industristandarder.

Almindelige typer accepttest omfatter følgende:

- Brugeraccepttest
- Driftsaccepttest
- Kontraktuelle og regulative accepttest
- Alfa- og betatest

De er beskrevet i de følgende fire delafsnit.

Brugeraccepttest (UAT)
Accepttest af systemet fokuserer typisk på at validere systemets egnethed for de tiltænkte brugere i et ægte eller simuleret driftsmiljø. Hovedmålsætningen er at opbygge tillid til, at brugerne kan anvende systemet til opfylde deres behov, leve op til kravene og udføre forretningsprocesser med minimale udfordringer, omkostninger og risici.

Driftsaccepttest (OAT)
Accepttest af systemet af drifts- og systemadministratorer udføres generelt i et (simuleret) produktionsmiljø. Testen fokuserer på driftsmæssige aspekter og kan omfatte:

- Test af backup og restore
- Installation, afinstallation og opgradering
- Disaster Recovery
- Brugerhåndtering
- Vedligeholdelsesopgaver

- Dataloads og migrationsopgaver
- Tjekker for sikkerhedsproblemer
- Performancetest

Den driftsmæssige accepttest har som hovedformål er at oparbejde tillid til at operatørerne eller systemadministratorerne kan holde systemet godt kørende for brugerne i driftsmiljøet eller selv under enestående eller besværlige betingelser.

Kontraktuel eller regulativ accepttest
Kontraktuelle accepttest udføres i henhold til en kontrakts acceptkriterier for at producere specieltudviklet software. Acceptkriterierne bør defineres, når parterne vedtager aftalen. Kontraktuelle accepttests udføres ofte af brugerne eller af uafhængige testere.
Regulativ accepttest udføres i henhold til alle regler, som de skal følge, så som reguleringer, love eller sikkerhedsregler. Regulativ accepttest udføres ofte af brugere eller af uafhængige testere, nogle gange hvor reglerne bevidnes eller revideres af kontrolorganer.
De kontraktuelle og regulative accepttest hovedmålsætninger er at oparbejde tillid til, at kontrakter og regler følges.

Alfa- og betatest
Alfa- og betatest anvendes typisk af udviklerne bag kommerciel standardsoftware (COTS), der ønsker feedback fra potentielle eller eksisterende brugere, kunder og/eller operatører, inden softwareproduktet lanceres på markedet. Alfatest udføres hos den udviklende organisation, ikke af udviklerholdet, men af de potentielle eller eksisterende kunder og/eller operatører eller et uafhængigt testteam. Betatest udføres af potentielle eller eksisterende kunder og/eller operatører på egne steder. Betatest kan udføres efter alfatest eller kan udføres uden nogen foregående alfatest har fundet sted.
Et af alfa- og betatestenes formål er at opbygge tillid blandt potentielle eller eksisterende kunder og/eller operatører til at de kan anvende systemet under normale, hverdagssituationer og i de driftsmæssige miljøer for at opnå deres målsætninger med minimale udfordringer, omkostninger og risici. En anden målsætning kan være at finde defekter, der er relateret til de betingelser og miljøer, som systemet skal anvendes i, særligt hvis disse betingelser og miljøer er svære at gentage for udviklingsteamet.

Testgrundlag
Eksempler på arbejdsprodukter, der kan anvendes som testgrundlag i enhver form for accepttests omfatter:

- Forretningsprocesser
- Bruger- eller forretningskrav
- Regulativer, kontrakter eller industristandarder
- Usecases og/eller userstory
- Systemkrav
- System- eller brugerdokumentation
- Installationsprocedurer
- Risikoanalyserapporter

Derudover kan et eller flere arbejdsprodukter anvendes som testgrundlag til at udlede testgrundlag for drifts-accepttest:

- Backup- og restoreprocedurer
- Disaster Recovery procedurer
- Ikke-funktionelle krav
- Driftsdokumentation
- Implementerings- og installationsinstruktioner
- Performancemål

- Databasepakker
- Sikkerhedsstandarder eller -regler

Typiske testobjekter
Typiske testobjekter for enhver form for accepttests omfatter:

- Systemer i test
- Systemkonfiguration og konfigurationsdata
- Forretningsprocedurer for et fuldt integreret system
- Restoresystemer og backup-facilitet (for forretningskontinuitet og Disaster Recovery tests)
- Drifts- og vedligeholdelsesprocesser
- Formularer
- Rapporter
- Eksisterende og konverterede data

Typiske defekter og afvigelser
Eksempler på typiske defekter for alle former for accepttest omfatter:

- Systemarbejdsgange lever ikke op til forretnings- eller brugerkrav
- Forretningens regler er ikke korrekt implementeret
- Systemet lever ikke op til kontraktuelle eller regulative krav
- Ikke-funktionelle, så som sårbarheder, utilstrækkeligt effektivitetsniveau for performance under høj belastning eller unormal drift på understøttet platform

Særlige tilgange og ansvarsområder
Accepttest er ofte kundens, forretningsrepræsentanter, product owners eller systemoperatørens ansvar, og andre interessenter kan også være involveret.
Accepttest anses ofte som det sidste testniveau i en sekventiel udviklingscyklus, men de kan også finde sted på andre tidspunkter f.eks.:

- Accepttest af COTS-softwareprodukter kan findes sted, når det er installeret eller integreret
- Accepttest af nye funktionelle udvidelser kan finde sted inden systemtesten

I iterativ udvikling kan projektteams anvende forskellige former for accepttest i løbet af og i slutningen af hver iteration, så som de, der har fokus på at verificere en ny funktion over for dens acceptkriterier, og de, der fokuserer på at validere, at nye funktioner lever op til brugernes behov. Derudover kan alfa- og betatest finde sted enten i slutningen af hver iteration, efter hver iteration er færdiggjort, eller efter en serie af iterationer. Brugeraccepttest, driftsaccepttest, regulativ accepttest og kontraktuel accepttest kan også finde sted enten tæt på hver iteration, efter færdiggørelse af hver iteration eller efter en serie af iterationer.

2.3 Testtyper

En testtype er en gruppe testaktiviteter, der sigter efter at teste et softwaresystems specifikke egenskaber eller en del af et system, baseret på et specifikt testformål. Sådanne målsætninger kan omfatte:

- Evaluere funktionelle kvalitetsegenskaber, så som fuldstændighed, korrekthed og egnethed
- Evaluere ikke-funktionelle kvalitetsegenskaber, så som performanceeffektivitet, sikkerhed, kompatibilitet og brugervenlighed
- Evaluere, om komponentens eller systemets struktur eller arkitektur er korrekt, komplet og følger specifikationerne

- Evaluere ændringernes effekt f.eks. for at bekræfte, at defekterne er blevet udbedret (gentest) og at søge efter utilsigtede ændringer i adfærd som resultat af ændringer i software eller miljø (regressionstest)

2.3.1 Funktionel test

Funktionelle tests af et system omfatter tests, der evaluerer funktioner, som systemet skal kunne udføre. Funktionelle krav kan beskrives i arbejdsprodukter, så som forretningskravenes specifikationer, epics, user-stories, usecases eller funktionelle specifikationer eller de kan være udokumenterede. Funktionerne beskriver, "hvad" systemet skal kunne.

Funktionel test bør udføres på alle testniveauer (f.eks. kan test af komponenter baseres på en komponentspecifikation), selvom fokusset er anderledes på hvert niveau (se afsnit 2.2).

Funktionel test betragter softwarens adfærd, så black-box teknikker kan anvendes til at udlede testbetingelser og testcases for komponentens eller systemets funktionalitet (se afsnit 4.2).

Grundigheden af den funktionelle test kan måles igennem den funktionelle dækning. Funktionel dækning er det omfang, hvorved en funktionalitet er blevet udøvet af tests og udtrykkes som en procentdel af den type af element, der dækkes. F.eks. kan man ved at anvende sporbarhed mellem tests og funktionelle krav udregne, hvilken procentdel af disse krav, der undersøges med testen, og potentielt set identificere huller i dækningen.

Funktionelt testdesign og afvikling kan indebære særlige færdigheder eller viden, så som viden om et særligt forretningsproblem, som softwaren løser (f.eks. geologiske modelleringssoftware til olie- og gasindustrien).

2.3.2 Ikke-funktionel test

Ikke-funktionel test af et system vurderer systemet og softwarens egenskaber, så som brugervenlighed, performanceeffektivitet eller sikkerhed. Se ISO Standard (ISO/IEC 25010) for klassifikation af softwareproduktets kvalitetsegenskaber. Ikke-funktionel test er at teste, "hvor godt" systemet håndterer sine opgaver.

I modsætning til den almen opfattelse, kan og bør ikke-funktionelle tests ofte udføres på alle testniveauer, og så tidligt som muligt. Hvis ikke-funktionelle defekter opdages sent, kan det udgøre stor risiko for at projektet ikke bliver vellykket.

Man kan anvende black-box teknikker til at udlede testbetingelser og testcases for ikke-funktionelle tests (se afsnit 4.2). F.eks. kan grænseværdianalyse anvendes til at definere stressbetingelser for performancetest.

Grundigheden ved ikke-funktionel test kan måles ved den ikke-funktionelle dækning. Ikke-funktionel dækning er det omfang, som en type af ikke-funktionelle elementer er blevet udøvet i testen og udtrykkes som en procentdel af den type elementer, der dækkes. F.eks. ved at anvende sporbarhed mellem tests og understøttede enheder til mobilapplikationer, kan procentdelen af enheder, der behandles i kompatibilitetstests udregnes, og man kan potentielt set identificere huller i dækningen.

Ikke-funktionel testdesign og afvikling kan kræve særlige evner eller viden, så som viden om indbyggede designsvagheder eller teknologiske svagheder (f.eks. sikkerhedsproblemer forbundet med specifikke programmeringssprog) eller den specifikke brugerbase (f.eks. personkredsen af brugere af styringssystemer i sundhedsplejefaciliteter).

Se ISTQB® CTAL-TA Certified Tester Advanced Level Test Analyst syllabus, ISTQB® CTAL-TTA Certified Tester Advanced Level Technical Test Analyst syllabus, ISTQB® CTAL-SEC Certified Tester Advanced Level Security Tester syllabus og andre ISTQB® specialistmoduler for flere detaljer vedrørende test af ikke-funktionelle kvalitetsegenskaber.

2.3.3 White-box test

White-box tests udleder tests baseret på systemets interne struktur eller implementering. Den interne struktur kan inkludere kode, arkitektur, arbejdsgange og/eller dataflows i systemet (se afsnit 4.3).

Grundigheden i white-box test kan måles igennem den strukturelle dækning. Den strukturelle dækning er det omfang, hvorved en type af strukturelle elementer er blevet behandlet i testene og udtrykkes som en procentdel af den type elementer, der dækkes.

På komponenttestniveau er kodedækningen baseret på procentdelen af komponentkode, der er blevet testet og kan måles i forhold til forskellige aspekter af koden (dækningselementer), så som procentdelen af eksekverbare instruktioner testet i komponenten eller procentdelen af testede beslutningsresultater. Denne type dækning kaldes samlet set for kodedækning. På komponentintegrationstestniveau kan white-box tests være baseret på systemets arkitektur, så som grænseflader mellem komponenter, og strukturel dækning kan måles i forhold til procentdelen af grænseflader afviklet i test.

White-box testdesign og afvikling kan indebære særlige evner eller viden, så som hvordan koden er opbygget, hvordan dataene lagres (f.eks. ved at evaluere mulige data-basesøgninger), og hvordan man anvender dækningsværktøjer og på korrekt vis tolker deres resultater.

2.3.4 Forandringsrelaterede test

Når der foretages ændringer i et system, enten for at udbedre en defekt eller pga. en ny eller ændret funktionalitet, bør der udføres test for at bekræfte at ændringerne har udbedret defekten eller implementeret funktionalitet korrekt og ikke har forårsaget nogen uforudsete, ugunstige konsekvenser.

- Gentest: Efter en defekt bliver udbedret, kan softwaren blive testet med alle de testcases, der tidligere har fejlet pga. defekten, og som bør kunne afvikles i den nye softwareversion. Softwaren kan også blive testet med nye tests for at dække ændringer i koden. Som minimum skal trinene for at reproducere den afvigelse, der forårsagede defekten, afvikles igen i den nye software-version. Formålet med gentesten er at få bekræftet, om den oprindelige defekt er udbedret
- Regressionstest: Det er muligt, at en forandring i en del af koden, uanset om det er en udbedring eller anden type ændring, ved en fejltagelse kan påvirke andre dele af kodens adfærd, uanset om det er inden for samme komponent, i andre af samme systems komponenter eller endda i andre systemer. Ændringer kan omfatte ændringer i miljøet, så som en ny version af et operativsystem eller styringssystem til databaser. Sådanne utilsigtede bivirkninger kaldes for regressioner. Regressionstest indebærer at udføre test for at spore sådanne utilsigtede bivirkninger

Gentest og regressionstest udføres på alle testniveauer.

Særligt i iterative og inkrementelle udviklingscyklusser (f.eks. Agile), nye funktioner, ændringer af eksisterende funktioner og refaktorering af koden resulterer i hyppige ændringer i koden, hvilket også kræver ændringsrelateret test. Da systemet er under udvikling, er gentest og regressionstest yderst vigtige. Dette er særligt relevant for Internet of Things-systemer, hvor individuelle objekter (f.eks. enheder) hyppigt opdateres eller udskiftes.

Regressionstestsuites afvikles ofte og udvikles generelt langsomt, så regressionstest er en stærk kandidat til automatisering. Automatisering af disse test bør påbegyndes tidligt i et projekt (se kapitel 6).

2.3.5 Testtyper og testniveauer

Det er muligt at udføre alle testtyper, der er nævnt ovenfor, på ethvert testniveau. For at illustrere det, bliver der givet eksempler på funktionelle, ikke-funktionelle, white-box og ændringsrelaterede tests på tværs af alle testniveauer for en bankapplikation, begyndende med funktionelle test:

- I komponenttest er tests designet ud fra, hvordan komponenten skal udregne rentes rente
- I komponentintegrationstest er testene designet ud fra, hvordan kontooplysningerne indsamles af brugergrænsefladen og videresendes til forretningslogikken
- I systemtest er testene designet ud fra, hvordan kontohaverne kan ansøge om en kreditlinje på deres bankkonto
- I systemintegrationstest er testene designet ud fra, hvordan systemet anvender en ekstern microservice for at tjekke en kontohavers kreditscore
- I accepttest er testene designet ud fra, hvordan banken godkender eller afviser en kreditansøgning

Det følgende er eksempler på ikke-funktionelle test:

- I komponenttest er performancetest designet til at evaluere antallet af CPU-cyklusser påkrævet for at udføre en kompleks total renteudregning
- I komponentintegrationstest er sikkerhedstest designet til bufferoverflowsproblemer pga. data overleveret fra brugergrænsefladen til forretningslogikken
- I systemtest er flytbarhedstest designet til at tjekke, om præsentationslaget virker på alle understøttede browsere og mobilenheder
- I systemintegrationstest er pålidelighedstest designet til at evaluere systemets robusthed, hvis den microservice, der anvendes til kreditscoring, ikke svarer
- I accepttest er brugervenlighedstest designet til at evaluere tilgængeligheden af bankens kredit-behandlingsbrugerflade for handicappede

De følgende er eksempler på white-box test:

- I komponenttest er testene designet til at opnå fuldkommen instruktions- og beslutningsdækning (se afsnit 4.3) for alle komponenter, der udfører finansielle udregninger
- I komponentintegrationstest er testene designet til at afvikle, hvordan hver skærm i browserbrugerfladen overfører data til næste skærm og til forretningslogikken
- I systemtest er testene designet til at dække sekvenser af hjemmesider, der kan forekomme i en kreditgivningsapplikation
- I systemintegrationstest er testene designet til alle mulige anmodningstyper, der sendes til den microservice, der anvendes til kreditscoring
- I accepttest er testene designet til at dække alle understøttede filstrukturer for finansielle data og værdiomfang i overførsler mellem banker

Endelig er de følgende eksempler på ændringsrelaterede test:

- I komponenttest er automatiserede regressionstest bygget til hver komponent og inkluderet i continuous integration frameworket
- For komponentintegrationstest er testene designet til at følge udbedringen af grænsefladerelaterede defekter, da udbedringerne bliver tjekket i kodearkivet
- I systemtest bliver alle test i en given arbejdsgang på ny afviklet, hvis nogle skærmbilleder i arbejdsgangen ændres
- I systemintegrationstesten er testene i den app, der interagerer med microservice, der anvendes til kreditscoring, afviklet på ny, dagligt som en del af continuous integration af denne microservice
- I accepttest bliver alle tidligere fejlede test på ny afviklet, efter en defekt, fundet i accepttesten, udbedres

Selvom dette afsnit giver eksempler på enhver testtype på tværs af niveauer, er det ikke nødvendigt for al software at gennemgå alle tests, der findes på hvert niveau. Det er derimod vigtigt at udføre alle anvendelige testtyper på hvert niveau, særligt i de tidligste niveauer hvor testtypen forekommer.

2.4 Vedligeholdelsestests

Når først softwaren og systemerne implementeres i produktionsmiljøerne, skal de vedligeholdes. Forskellige typer af ændringer i den leverede software og systemer er næsten uundgåelige, enten for at udbedre defekter, der opdages i drift, for at tilføje ny funktionalitet eller for at slette eller ændre allerede leveret funktionalitet. Vedligeholdelse er også påkrævet for at forbedre komponentens eller systemets ikke-funktionelle systemegenskaber igennem dets levetid, særligt performanceeffektivitet, pålidelighed, sikkerhed og flytbarhed.

Når ændringer foretages som en del af vedligeholdelsen, bør vedligeholdelsestest udføres, både for at evaluere, hvor vellykket de foretagne ændringer er og for at tjekke for mulige bivirkninger (f.eks. regressioner) i de dele af systemet, som er uændret (hvilket typisk er størstedelen af systemet). Vedligeholdelse kan indebære planlagte releases og ikke-planlagte releases (hot fixes).

En vedligeholdelsesrelease kan kræve vedligeholdelsestest på flere testniveauer vha. flere testtyper baseret på dens omfang. Vedligeholdelsestestens omfang afhænger af:

- Ændringens risikograd f.eks. i hvilken grad, den ændrede del af softwaren kommunikerer med andre komponenter eller systemer
- Det eksisterende systems størrelse
- Ændringens størrelse

2.4.1 Triggere for vedligeholdelse

Der er adskillige årsager til, at softwarevedligeholdelse, og dermed også vedligeholdelsestest, finder sted både for planlagte og ikke-planlagte ændringer.

Vi kan klassificere de udløsende faktorer for vedligeholdelse således:

- Ændringer, for eksempel planlagte udvidelser (f.eks. releasebaseret), korrigerende eller nødændringer, ændringer af driftsmiljø (planlagte opgraderinger af operativsystem eller databaser), opgraderinger af COTS-software og patches af defekter og sårbarheder
- Migration, for eksempel fra en platform til en anden, hvilket kan kræve driftsmæssige tests både af det nye miljø og den ændrede software, eller tests af datakonvertering, når data fra en anden app, der skal migreres ind i et system undergår vedligeholdelse
- Pensionering, når en applikation tages ud af brug
    - Når en applikation eller et system pensioneres, kan det kræve test af datamigration eller arkivering, hvis der er krav om lang tids lagring af data
    - Test af gendannelses-/indhentningsprocedurer efter arkivering med data-opbevaringstider over lange perioder kan også være nødvendigt
    - Det kan være nødvendigt med regressionstest for at sikre, at den funktionalitet, der forbliver i brug, stadig virker

I Internet of Things-systemer kan der være grund til at udføre vedligeholdelsestests udløses ved introduktion af helt nye eller ændrede enheder som f.eks. hardwareenheder og softwaretjenester ind i det overordnede system. Vedligeholdelsestest i sådanne systemer lægger særlig fokus på integrationstest på forskellige niveauer (f.eks. netværksniveau og applikationsniveau) og på sikkerhedsaspekter, især mht. persondata.

2.4.2 Effektanalyse af vedligeholdelsen

Effektanalysen evaluerer de ændringer, der er udført i en vedligeholdelsesrelease for at identificere følgevirkninger, såvel som de forventede og mulige bivirkninger ved ændringen, og for at identificere områder i systemet, der påvirkes af ændringen. Effektanalysen kan også hjælpe med at identificere ændringens effekt på

eksisterende test. Bivirkningerne og de påvirkede områder i systemet skal testes for regressioner, muligvis efter at have opdateret eksisterende test, der er påvirket af ændringen.
Effektanalysen kan udføres, inden ændringen udføres, for at hjælpe med at beslutte, om ændringen skal gennemføres baseret på de potentielle følgevirkninger i andre dele af systemet.

Effektanalysen kan være problematisk, hvis:

- Specifikationer (f.eks. forretningskrav, userstories, arkitektur) er for gamle eller mangler
- Testcases ikke er dokumenteret eller er for gamle
- Der ikke har været tovejs sporbarhed mellem tests og testgrundlaget
- Værktøjsunderstøttelsen er svag eller ikke-eksisterende
- De involverede mennesker ikke har domæne- og/eller systemviden
- Man ikke har været tilstrækkeligt opmærksom på softwarens vedligeholdelsesegnethed under udviklingen

3. Statisk test — 135 minutter

Nøgleord

ad hoc review, tjeklistebaseret review, dynamisk test, formelt review, uformelt review, inspektion, perspektivbaseret læsning, review, rollebaseret review, scenariebaseret review, statisk analyse, statisk test, teknisk review, walk-through

Læringsmål for statisk test:

3.1 Grundlæggende principper for statisk test

FL-3.1.1 (K1) Kend de typer softwareprodukter, som kan undersøges med forskellige statiske testteknikker

FL-3.1.2 (K2) Brug eksempler til at beskrive værdien af statisk test

FL-3.1.3 (K2) Forklar forskellen mellem statiske og dynamiske teknikker med målsætninger, defekttyper, der skal identificeres, og de roller, som de teknikker spiller inden for softwarens livscyklus

3.2 Reviewproces

FL-3.2.1 (K2) Opsummer aktiviteterne i arbejdsproduktets reviewproces

FL-3.2.2 (K1) Beskriv de forskellige roller og ansvarsområder i et formelt review

FL-3.2.3 (K2) Forklar forskellene mellem de forskellige typer af reviews: uformelt review, walkthrough, teknisk review og inspektion

FL-3.2.4 (K3) Anvend en reviewteknik på et arbejdsprodukt for at finde defekter

FL-3.2.5 (K2) Forklar de faktorer, der bidrager til et vellykket review

3.1 Grundlæggende statisk test

I modsætning til dynamisk test, der kræver afvikling af den software, som testes, består statisk test af en manuel undersøgelse af arbejdsprodukter (dvs. reviews) eller værktøjsdrevet evaluering af koden eller andre arbejdsprodukter (dvs. statisk analyse). Begge typer af statisk test vurderer koden eller andre arbejdsprodukter, der testes, uden egentlig afvikling af koden eller det testede arbejdsprodukt.

Statiske analyser er vigtige for sikkerhedskritiske IT-systemer (f.eks. luftfartssoftware, medicoteknisk eller software til styring af atomkraftværker), men statiske analyser er også blevet vigtige og almindelige i andre sammenhænge. F.eks. er statiske analyser en vigtig del af sikkerhedstest. Statiske analyser er også ofte indbygget i automatiske software build- og delivery-værktøjer f.eks. i agil udvikling, continuous delivery og anden continuous integration.

3.1.1 Arbejdsprodukter, der kan undersøges i statisk test

Næsten alle arbejdsprodukter kan undersøges i statisk test (reviews og/eller statiske analyser) f.eks.:

- Specifikationer, herunder forretningskrav, funktionelle krav og sikkerhedskrav
- Epics, userstories og acceptkriterier
- Arkitektur- og designspecifikationer
- Kode
- Testware, herunder testplaner, testcases, testprocedurer og automatiserede testscripts
- Brugervejledninger
- Hjemmesider
- Kontrakter, projektplaner, tidsplaner og budgetter
- Konfigurations- og infrastruktur-setupinfrastruktursetup
- Modeller, f.eks. aktivitetsdiagrammer, der kan anvendes til modelbaseret test (se ISTQB® CTFL-MBT Certified Tester Foundation Level Model-Based Tester syllabus og Kramer 2016)

Reviews kan anvendes på ethvert arbejdsprodukt, som deltagerne kan læse og forstå. Statiske analyser kan anvendes effektivt til alle arbejdsprodukter med en formel struktur (typisk kode eller modeller), der findes et passende statisk analyseværktøj til. Statisk analyse kan endda anvendes med værktøjer, der evaluerer produkter, som er skrevet i naturligt sprog, så som krav (f.eks. at tjekke for stavefejl, grammatiske fejl og læsbarhed).

3.1.2 Fordele ved statiske tests

Statiske testteknikker rummer en række fordele. Hvis de udføres tidligt i softwareudviklingslivs-cyklussen, gør statiske test det muligt at finde defekter tidligt, inden de dynamiske test udføres (f.eks. i review af krav- eller designspecifikationerne, refinement af produkt-backlog osv.) Defekter, der findes tidligt, er ofte billigere at fjerne, end defekter, som findes sent i livscyklussen, især i sammenligning med defekter, der findes efter softwaren udsendes og sættes i drift. Det er næsten altid billigere for en organisation at anvende statiske testteknikker til at finde defekter, der kan rettes med det samme, i stedet for senere at anvende dynamiske tests til at finde defekter i testobjektet og derefter udbedre dem, især når man medregner omkostningerne ved at opdatere andre arbejdsprodukter og udføre gentest og regressionstest.

Yderligere fordele ved statisk test:

- At finde og udbedre defekter inden den dynamiske test udføres
- At opdage defekter, som ikke er nemme at finde ved dynamisk tests
- At forhindre defekter i design eller kodning ved at afdække uoverensstemmelser, tvetydigheder, selvmodsigelser, udeladelser, unøjagtighed og redundans i kravene
- At forøge produktiviteten (f.eks. pga. forbedret design og kode, der er nemmere at vedligeholde)

- At reducere tid og omkostninger til udvikling
- At reducere tid og omkostninger til test
- At reducere de samlede kvalitetsomkostninger gennem hele softwarens levetid pga. færre afvigelser senere i livscyklussen og efter levering til drift
- At forbedre kommunikation mellem teamets medlemmer, mens de deltager i reviews

3.1.3 Forskellene mellem statisk og dynamisk test

Statisk og dynamisk test kan have samme målsætninger (se afsnit 1.1.1), så som at give en vurdering af arbejdsprodukternes kvalitet og at identificere defekter så tidligt som muligt. Statisk og dynamisk test komplementerer hinanden ved at finde forskellige typer af defekter.

En af de primære forskelle er, at statisk test finder defekter i arbejdsprodukter direkte, i stedet for at identificere afvigelser, forårsaget af defekter, når softwaren afvikles. En defekt kan eksistere i et arbejdsprodukt i lang tid uden at forårsage afvigelser. Den sti, hvor defekten findes, kan være sjældent benyttet eller svær at nå frem til, så det er ikke nemt at konstruere og afvikle dynamiske tests, der kan afsløre den. Statiske tests kan finde sådanne defekter med mindre besvær.

En anden forskel er, at statiske tests kan anvendes til at forbedre arbejdsprodukternes uoverensstemmelser og interne kvalitet, hvorimod dynamiske tests typisk fokuserer på det eksternt synlige adfærd.

Til sammenligning med dynamisk test omfatter de typiske defekter, som er nemmere og billigere at finde og udbedre vha. statisk test:

- Kravdefekter (f.eks. uoverensstemmelser, tvetydigheder, selvmodsigelser, udeladelser, unøjagtigheder og redundans)
- Designdefekter (f.eks. utilstrækkelige algoritmer eller databasestrukturer, høj kobling, lav sammenhæng)
- Kodedefekter (f.eks. variable med udefinerede værdier, erklærede variabler, som aldrig bliver anvendt, utilgængelig kode, duplikeret kode)
- Afvigelser fra standarderne (f.eks. kode, der ikke følger kodestandarderne)
- Ukorrekte grænsefladespecifikationer (hvis f.eks. det kaldende system anvender andre enheder end det kaldte system)
- Sikkerhedsbrister (f.eks. modtagelig over for bufferoverløb)
- Huller eller unøjagtigheder i testgrundlagets sporbarhed eller dækning (f.eks. manglende test for et acceptkriterium)

De fleste typer af vedligeholdelsesdefekter kan kun findes med statiske tests (f.eks. forkert modularisering, komponenternes ringe genbrugelighed, kode, der er svær at analysere og modificere uden at introducere nye defekter).

3.2 Reviewproces

Reviews kan være alt fra uformelle til formelle. Uformelle reviews er kendetegnet ved, at de ikke følger en foruddefineret proces og ikke har krav om formelt dokumenteret output. Formelle reviews er kendetegnet ved team-deltagelse, dokumenterede resultater af reviewet og dokumentation af processen, der udfører reviewet. Reviewprocessens grad af formalitet er relateret til faktorer som modellen for softwareudviklingslivscyklussen, udviklingsprocessens modenhed, det reviewede produkts kompleksitet, lovmæssige eller regulative krav og/eller behovet for revisionsspor.

Reviewets fokus afhænger af de målsætninger, der på forhånd er aftalt for reviewet (f.eks. at finde defekter, opnå forståelse, undervise deltagere så som testere og nye medlemmer af teamet eller diskutere og afgøre ved almen enighed).

ISO-standard (ISO/IEC 20246) indeholder dybdegående beskrivelser af reviewprocessen for arbejdsprodukter, herunder roller og reviewteknikker.

3.2.1 Reviewprocessen for arbejdsprodukter

Reviewprocessen udgøres af de følgende hovedaktiviteter:

Planlægning

- Definition af omfang, reviewets formål, hvilke dokumenter og dele af dokumenterne, som skal reviewes og de kvalitetskriterier, som skal evalueres
- Vurdering af indsats og tidsramme
- Identifikation af reviewkarakteristika, f.eks. reviewtype med roller, aktiviteter og tjeklister
- Udvalg af deltagere til reviewet og fastlæggelse af deres roller
- Definition af start- og slutkriterier for formelle reviewtyper (f.eks. inspektion)
- Kontrol af, at startkriterierne er opfyldt (for formelle reviewtyper)

Igangsætte reviewet

- Distribution af arbejdsprodukter (via fysiske eller elektroniske midler) og andet materiale, f.eks. logformularer, tjeklister og relaterede arbejdsprodukter
- Forklaring af omfang, målsætninger, proces, roller og arbejdsprodukter for deltagerne
- At svare på alle spørgsmål, som deltagerne kan have til reviewet

Individuelle reviews (dvs. individuel forberedelse)

- Review hele eller dele af arbejdsproduktet
- Bemærk potentielle defekter, anbefalinger og spørgsmål

Kommunikation og analyse af problemer

- Kommunikation af de identificerede defekter (f.eks. i et review-møde)
- Analyse af potentielle defekter og tildeling af ejerskab og status for dem
- Evaluering og dokumentation af kvalitetsegenskaber
- At evaluere, hvad reviewet har opdaget holdt op mod slutkriterierne for at tage en beslutning (afvise; større ændringer behøves; accepter, muligt med mindre ændringer)

Udbedring og rapportering

- At skrive defektrapporter om de fund, der kræver ændringer i et arbejdsprodukt
- At udbedre fundne defekter i arbejdsprodukter, der er blevet reviewet (typisk udført af forfatteren)
- At kommunikere defekterne til den rette person eller team (når de findes i et arbejdsprodukt, relateret til det arbejdsprodukt, som er blevet reviewet)
- Dokumentere den opdaterede defektstatus (i formelle reviews) potentielt med den oprindelige kommentators samtykke
- At indsamle metrikker (til mere formelle reviewtyper)
- At tjekke at slutkriterierne følges (for mere formelle reviewtyper)
- At godkende arbejdsproduktet, når slutkriterierne er opnået

Resultaterne i et review af et arbejdsprodukt kan variere afhængigt af reviewtypen og graden af formalitet som beskrevet i afsnit 3.2.3.

3.2.2 Roller og ansvarsområder i formelle reviews

Et typisk formelt review omfatter rollerne nedenfor:

Forfatter

- Har lavet det arbejdsprodukt, der reviewes
- Udbedrer defekter i arbejdsproduktet som bliver reviewet (om nødvendigt)

Ledelse

- Er ansvarlig for planlægning af review
- Foretager beslutninger om udførelse af reviews
- Tildeler arbejdskraft, budget og tid
- Overvåger løbende omkostningseffektiviteten
- Tager beslutninger om kontrol i tilfældet af utilstrækkelige resultater

Facilitator (kaldes ofte moderator)

- Sørger for at reviewmøderne gennemføres på effektiv vis (når de holdes)
- Formidler oplysninger mellem forskellige synspunkter, om nødvendigt
- Er ofte den, der er afgørende for, at reviewet bliver vellykket

Reviewleder

- Har det overordnede ansvar for reviewet
- Afgør, hvem der deltager, og planlægger, hvornår og hvor det finder sted

Reviewere

- Kan være eksperter inden for området, personer, der arbejder på projektet, interessenter med interesse i arbejdsproduktet og/eller personer med særlig, teknisk eller erhvervsmæssig baggrund
- Opdager potentielle defekter i det arbejdsprodukt, der reviewes
- Kan repræsentere forskellige perspektiver (f.eks. tester, udvikler, bruger, operatør, forretnings-analytiker, brugervenlighedsekspert osv.)

Referent (eller notatskriver)

- Samler de potentielle defekter, der bliver fundet
- Nedskriver de nye potentielle defekter, åbne punkter og beslutninger foretaget under review-mødet (når det afholdes)

I nogle reviewtyper kan en person spille mere end en rolle, og de handlinger, som knyttes til hver rolle, kan også variere baseret på reviewtypen. Derudover er der ofte ikke behov for en referent, når der introduceres værktøjer, der understøtter reviewprocessen. Dette er især tilfældet, når det drejer sig om at logge defekter, åbne punkter og beslutninger, der vedtages undervejs.

Dertil kan der også være behov for mere detaljerede roller som beskrevet i ISO-standard (ISO/IEC 20246)

3.2.3 Reviewtyper

Selvom reviews kan anvendes til forskellige formål, er et af hovedmålsætninger at afdække defekter. Alle reviewtyper kan hjælpe til at finde defekter, og den valgte reviewtype bør baseres på de behov, der opstår i projektet, de tilgængelige resurser, produkttype og risici, forretningsdomæne og forretningskultur mv.

Et enkelt arbejdsprodukt kan gennemgå flere en én type review. Rækkefølgen af reviewtyper vil afhænge af de rammer som de skal udføres inden for. For eksempel kan det være brugbart at udfører et uformelt review før et tekniskreview for at sikre at arbejdsproduktet er klar til dette.

De reviewtyper der er oplistet herunder, kan alle blive udført som peer review, det vil sige udført af andre som er kvalificeret til at udføre samme arbejdsopgave som forfatteren.

Defekter fundet under et review kan variere i type dette afhængig specielt af det arbejdsprodukt som er under review (se sektion 3.1.3 for eksempler på defekter der kan blive fundet under et review og Gilb 1993 for information om formel inspektion).

Reviews kan klassificeres i henhold til forskellige egenskaber. Det følgende er en liste over de fire mest almindelige typer af reviews og deres tilknyttede egenskaber.

Uformelt review (f.eks. kollegagennemsyn, pairing, par-reviews)

- Hovedformål: finde potentielle defekter
- Yderligere mulige formål: generere nye idéer eller løsninger, hurtigt at løse mindre problemer
- Ikke baseret på formelle (dokumenterede) processer
- Behøver ikke at indebære review-møde
- Kan udføres af forfatterens kollega (kollegagennemsyn) eller af flere personer
- Resultaterne kan dokumenteres
- Anvendeligheden afhænger af reviewerne
- Om tjeklister anvendes er valgfrit
- Anvendes meget ofte i agil udvikling

Walkthrough

- Hovedformål: finde defekter, forbedre softwareproduktet, overveje alternative implementeringer, evaluere overensstemmelse i forhold til standarder og specifikationer
- Yderligere mulige formål: udveksle idéer om teknikker eller stilvarianter, træning af deltagere, opnå enighed
- Individuel forberedelse inden reviewmødet er valgfrit
- Reviewmødet ledes typisk af arbejdsproduktets skaber
- Referenten er obligatorisk
- Anvendelse af tjeklister er valgfrit
- Kan finde sted som scenarier, dry runs eller simuleringer
- Potentielle defektlogs og reviewrapporter udarbejdes
- Kan i praksis variere fra ganske uformelt til meget formelt

Teknisk review

- Hovedformål: opnå enighed, finde potentielle defekter
- Andre mulige formål: evaluere kvaliteten og opbygge tillid til arbejdsprodukter, generere nye idéer, motivere og gøre det muligt for skabere at forbedre fremtidige arbejdsprodukter, overveje alternative implementeringer

- Reviewere bør være teknisk ligestillet med forfatteren og være tekniske eksperter inden for samme eller andre discipliner
- Individuel forberedelse inden reviewmødet er påkrævet
- Reviewmødet er valgfrit, og skal ideelt set ledes af en trænet facilitator (helst ikke forfatteren)
- Det er obligatorisk med en referent. Det skal helst ikke være forfatteren
- Anvendelse af tjeklister er valgfrit
- Mødets resultat er en defektlog og reviewrapporter

Inspektion

- Hovedformål: At opdage potentielle defekter, at evaluere kvaliteten og opbygge tillid til arbejdsproduktet, at forhindre lignende defekter i opstå i fremtiden og analyse af rodårsager
- Yderligere mulige formål: At motivere og gøre det muligt for forfattere at forbedre fremtidige arbejdsprodukter og softwareudviklingsprocessen, at opnå enighed
- Følger en foruddefineret proces med formelle, dokumenterede outputs, baseret på regler og tjeklister
- Anvender klart definerede roller som de, der er specificeret i afsnit 3.2.2, der er obligatoriske og indebærer en dedikeret læser (der læser arbejdsproduktet højt, og ofte med deres egne ord, under reviewmødet)
- Det er nødvendigt med individuel forberedelse inden reviewmødet
- Reviewere er enten på lige niveau med forfatteren eller eksperter i andre discipliner, der er relevant for arbejdsproduktet
- Der anvendes specifikke start- og slutkriterier
- Det er obligatorisk at skrive referat
- Reviewmødet ledes af en trænet facilitator (ikke af forfatteren)
- Forfatteren kan ikke fungere som reviewleder, læser eller referent
- Der skal udarbejdes potentielle defektlogge og reviewrapporter
- Data samles og anvendes til at forbedre hele softwareudviklingsprocessen, herunder inspektionsprocessen

3.2.4 Anvendelse af reviewteknikker

Der findes et antal reviewteknikker, der kan anvendes i løbet af et individuelt reviewaktivitet (dvs. individuel forberedelse) til at afdække defekter. Disse teknikker kan anvendes på tværs af de reviewtyper, der står beskrevet ovenfor. Teknikkernes effektivitet kan afhænge af den reviewtype, der anvendes. Eksempler på forskellige, individuelle reviewteknikker til forskellige reviewtyper beskrives nedenfor.

Ad hoc

I et ad hoc-review gives reviewerne enten lidt eller ingen vejledning om, hvordan opgaven skal udføres. Reviewere læser ofte arbejdsproduktet sekventielt, og identificerer og dokumenterer problemerne, når de opdager dem. Ad hoc-reviews er en ofte anvendt teknik, der kun kræver en smule forberedelse. Denne teknik er meget afhængig af reviewerens evner og kan lede til, at mange ens problemer bliver rapporteret af forskellige reviewere.

Tjeklistebaseret

Et tjeklistebaseret review er en systematisk teknik, hvorved revieweren finder problemer baseret på tjeklister, der er bliver delt ud ved begyndelsen af reviewet (f.eks. af facilitatoren). En review-tjekliste består af et sæt spørgsmål, der er baseret på potentielle defekter, som kan være udledt af erfaringer. Tjeklister bør specifikt gælde den type arbejdsprodukt, der reviewes, og bør ofte vedligeholdes for at dække problemtyper, som undveg tidligere reviews. Hovedfordelen ved den tjeklistebaserede teknik er den systematiske dækning af typiske defekttyper. Man bør være forsigtig med ikke blot at følge tjeklisten i de individuelle reviews, men også lede efter defekter, som ikke står i tjeklisten.

Scenarier og dry runs
I scenariebaserede reviews, gives reviewerne strukturerede retningslinjer for, hvordan man skal gennemlæse arbejdsproduktet. En scenariebaseret tilgang støtter revieweren i udførslen af "dry runs" af arbejdsproduktet, baseret på den forventede anvendelse af arbejdsproduktet (hvis arbejdsproduktet er dokumenteret i et passende format som usecases). Disse scenarier giver reviewerne bedre retningslinjer for at identificere defekttyper end simple enkelte på en tjekliste. Ligesom i tjeklistebaserede reviews for ikke at overse andre typer defekter (f.eks. manglende egenskaber), bør reviewere ikke være bundet til det dokumenterede scenarie.

Perspektivbaseret
I perspektivbaseret læsning skal reviewere tage forskellige interessenters synspunkter i individuelle reviews ligesom i rollebaserede reviews. Typiske interessentsynspunkter omfatter slutbrugere, marketing, designer, tester og operatører. Når man anvender forskellige interessentsynspunkter, fører det til mere dybdegående individuelle reviews med færre gentagelser af problemer på tværs af reviewere.

Derudover kræver perspektivbaseret læsning også, at revieweren forsøger at anvende det reviewede arbejdsprodukt til at generere det produkt, der skal udledes fra det. F.eks. bør en teste forsøge at genere et udkast til en accepttest, hvis han eller hun udfører en perspektivbaseret læsning af en kravspecifikation for at se, om alle de nødvendige oplysninger findes. Derudover forventes det, at der anvendes tjeklister i perspektiv-baseret læsning.

Empiriske studier har vist, at perspektivbaseret læsning er den mest effektive, generelle teknik til review af krav og tekniske arbejdsprodukter. En vigtig succesfaktor er at inkludere og afveje de forskelle interessenters synspunkterne på passende vis og baseret på risici. Se Shul 2000 for detaljer om perspektivbaseret læsning og Sauer 2000 for mere om de forskellige review-teknikker effektivitet.

Rollebaseret
Et rollebaseret review er en teknik, hvori reviewerens rolle er at evaluere arbejdsproduktet fra interessentens perspektiv. Typiske roller omfatter specifikke slutbrugertyper (erfarne, uerfarne, ældre, børn, osv.) og specifikke roller i organisationen (brugeradministrator, systemadministrator, performancetester, osv.). De samme principper anvendes som ved perspektivbaseret læsning da rollerne er ens.

3.2.5 Succesfaktorer for reviews

For at gennemføre et vellykket review, skal man overveje de passende typer af reviews og anvendte teknikker. Derudover findes der en række faktorer, der kan påvirke reviewets udfald.

De organisatoriske succesfaktorer for reviews omfatter:

- Hvert review har klare målsætninger, der defineres ved planlægningen af reviewet og anvendes som målbart slutkriterier
- Der anvendes reviewtyper, som er i stand til at opnå målsætningerne, og som passer til de typer og niveauer af software arbejdsprodukter og deltager
- Alle de reviewteknikker, der anvendes, så som tjeklistebaseret eller rollebaseret reviews passer til effektiv identifikation af defekter i det reviewede arbejdsprodukt
- Alle anvendte tjeklister tager højde for hovedrisici og er opdaterede
- Større dokumenter skal skrives og reviewes i mindre dele, så kvalitetskontrollen udføres ved at give skaberne tidlig og hyppig feedback om defekter
- Deltagerne skal have tilstrækkelig tid til at forberede sig
- Reviews planlægges med tilstrækkeligt varsel
- Ledelsen støtter reviewprocessen (f.eks. ved at indarbejde tilstrækkeligt med tid til review-aktiviteterne i projektplanerne)
- Review er integreret i forretningens kvalitets- og/eller testpolitik

Personrelaterede succesfaktorer for reviews omfatter:

- De rigtige folk deltager for at opfylde reviewets målsætninger, f.eks. folk med forskellige evner eller perspektiver, der kan anvende dokumentet som input til arbejdet
- Testere skal anses som værdifulde reviewere, der kan bidrage til reviewet og lære om arbejdsproduktet, hvilket gør det i stand til at forberede mere effektive test og at forberede de test tidligere
- Deltagerne skal have afsat eller tildelt en passende mængde tid og opmærksomhed til detaljerne
- Reviews udføres på små dele, så reviewerne ikke mister koncentrationen i løbet af de individuelle reviews og/eller review-møder (når de holdes)
- De defekter, der findes, anerkendes, betragtes og håndteres objektivt
- Mødet skal håndteres ordentligt, så deltagerne ikke mener, at de spilder deres værdifulde tid
- Reviewet skal udføres i en tillidsfuld atmosfære, og udfaldet bliver ikke anvendt til at evaluere deltagerne
- Deltagerne skal undgå kropssprog og adfærd, der kan indikere kedsomhed, irritation eller fjendtlighed mod andre deltagere
- Der skal ydes tilstrækkelig træning, særligt i mere formelle reviewtyper som inspektion
- Der skal fremmes en kultur med læring og forbedring

Se Gilb 1993, Wiegers 2002 og Veenendaal 2004 for mere om vellykkede reviews.

4. Testteknikker — 330 minutter

Nøgleord

beslutningsdækning, beslutningstabeltest, black-box testteknik, dækning, erfaringsbaseret testteknik, fejlgætning, grænseværdianalyse, instruktionsdækning, testteknik, tilstandsovergangstest, tjeklistebaseret test, udforskende test, usecasetest, white-box testteknik, ækvivalenspartitionering

Læringsmål for testteknikker

4.1 Kategorier af testteknikker

FL-4.1.1 (K2) Forklar karakteristika, fællestræk og forskelligheder mellem black-box testteknikker, white-box testteknikker og erfaringsbaserede testteknikker

4.2 Black-box testteknikker

FL-4.2.1 (K3) Anvend ækvivalenspartitionering for at udlede testcases fra givne krav
FL-4.2.2 (K3) Anvend grænseværdianalyse for at udlede testcases fra givne krav
FL-4.2.3 (K3) Anvend beslutningstabeltest for at udlede testcases fra givne krav
FL-4.2.4 (K3) Anvend tilstandsovergangstest for at udlede testcases fra givne krav
FL-4.2.5 (K2) Forklar, hvordan man udleder testcases fra en usecase

4.3 White-box testteknikker

FL-4.3.1 (K2) Forklar instruktionsdækning
FL-4.3.2 (K2) Forklar beslutningsdækning
FL-4.3.3 (K2) Forklar værdien af instruktions- og beslutningsdækning

4.4 Erfaringsbaserede testteknikker

FL-4.4.1 (K2) Forklar fejlgætning
FL-4.4.2 (K2) Forklar udforskende test
FL-4.4.3 (K2) Forklar tjeklistebaseret test

4.1 Kategorier af testteknikker

Formålet med testteknikker er at hjælpe med at identificere testbetingelser, testcases og testdata.

Valget af, hvilket testteknikker, der skal anvendes, afhænger af en række faktorer, herunder følgende:

- Komponentens og systemets kompleksitet
- Regulative standarder
- Kunde- eller kontraktlige krav
- Risikoniveauer og typer
- Tilgængelig dokumentation
- Testerens viden og færdigheder
- Tilgængelige værktøjer
- Tid og budget
- Softwareudviklingens livscyklusmodel
- Typer af defekter, der forventes i komponenten eller systemet

Nogle teknikker er mere anvendelige i bestemte situationer og på bestemte testniveauer; andre kan anvendes på alle testniveauer. Testere anvender i almindelighed en blanding af forskellige testteknikker, når de udarbejder testcases for at opnå de bedste resultater for indsatsen.

De anvendte testteknikker i testanalyse-, testdesigns- og testimplementeringsaktiviteterne kan gå fra meget uformelle til meget formelle. Et passende formalitetsniveau afhænger af konteksten, herunder testens og udviklingsprocessernes modenhed, tidsfrister, sikkerhed og regulative krav, de involverede personers viden og færdigheder og den livscyklusmodel, der følges i softwareudviklingen.

4.1.1 Kategorier af testteknikker og deres karakteristika

I denne syllabus bliver testteknikkerne klassificeret som black-box, white-box eller erfaringsbaseret test.

Black-box testteknikker (der også kaldes adfærdsmæssige eller adfærdsbaserede teknikker) baseres på en analyse af et passendes testgrundlag (f.eks. formelle kravdokumenter, specifikationer, usecases, userstories eller forretningsprocesser). Disse teknikker kan både anvendes i funktionel og ikke-funktionel test. Black-box testteknikker fokuserer på testobjektets input og output uden at forholde sig til testobjektets interne struktur.

White-box testteknikker (der også kaldes strukturelle eller strukturbaserede teknikker) baseres på en analyse af arkitekturen, detaljeret design, intern struktur eller testobjektets kode. I modsætning til black-box fokuserer white-box på struktur og processer i testobjektet.

Erfaringsbaserede testteknikker understøtter udviklernes, testernes og brugernes erfaringer med design, implementering og afvikling af test. Disse teknikker kombineres ofte med black-box og white-box testteknikker.

Black-box testteknikkernes almene karakteristika omfatter:

- Testbetingelser, testcases og testdata udledes af et testgrundlag, som kan bestå af krav til softwaren, specifikationer for usecases og userstories
- Testcases kan anvendes til at finde afvigelser mellem krav og implementeringen af dem
- Man kan måle testdækningen på grundlag af, hvad der testes i testgrundlaget og de teknikker, der anvendes på testgrundlaget

White-box testteknikkernes almene karakteristika omfatter:

- Testbetingelser, testcases og testdata afledes af et testgrundlag, der kan indeholde kode, softwarearkitektur, detaljeret design og andre kilder til softwarens struktur
- Testdækningen måles på grundlag af, hvad der testes i den valgte struktur (f.eks. kode eller grænseflader) og den teknik der er brugt på testgrundlaget

De erfaringsbaserede testteknikkers almene karakteristika omfatter:

- Testbetingelser, testcases og testdata udledes af et testgrundlag, der bygger på testernes, udviklernes, brugernes og andre interessenters viden og erfaring

Viden og erfaring om softwarens forventede anvendelse, dens miljø, sandsynlige defekter og fordelingen deraf.

Den internationale standard (ISO/IEC/IEEE 29119-4) indeholder beskrivelser af testteknikker og deres tilsvarende dækningstyper (se Craig 2002 og Copeland 2004 for flere oplysninger om teknikkerne).

4.2 Black-box testteknikker

4.2.1 Ækvivalenspartitionering

Ækvivalenspartitionering opdeler data i partitioner (også kendt som ækvivalensklasser) på en sådan måde, at alle dele af en givet partition kan forventes at blive behandlet på samme måde (se Kaner 2013 og Jorgensen 2014). Der findes ækvivalenspartitioner for både gyldige og ugyldige værdier.

- Gyldige værdier er værdier, der bør blive accepteret af komponenten eller systemet. En ækvivalenspartition, der indeholder gyldige værdier, kaldes for en "gyldig ækvivalenspartition"
- Ugyldige værdier er værdier, der bør afvises af komponenten eller systemet. En ækvivalenspartition, der indeholder ugyldige værdier kaldes for en "ugyldig ækvivalenspartition"
- Partitioner kan identificeres på alle dataelementer, der er relateret til testobjektet, inklusiv input, output, interne værdier, tidsrelaterede værdier (f.eks. før eller efter et event) og til brugerfladens parametre (f.eks. integrerede komponenter, der testes i løbet af integrationstesten)
- Enhver partition kan opdeles i underpartitioner, hvis det er nødvendigt
- Enhver værdi tilhører én og kun én ækvivalenspartition
- Når ugyldige ækvivalenspartitioner anvendes i testcases, skal de testes hver for sig, dvs. ikke i kombination med andre ugyldige ækvivalenspartitioner. Det er for at få afdækket alle afvigelser. Afvigelser kan skjule sig, når adskillige afvigelser finder sted på samme tid, og kun ét tilfælde er synligt

For at opnå 100% dækning med denne teknik, skal testcases dække alle identificerede partitioner (herunder ugyldige partitioner) ved at anvende mindst en værdi fra hver partition. Dækningen måles som antallet af ækvivalenspartitioner, der testes med mindst en værdi divideret med det samlede antal identificerede ækvivalenspartitioner. Skrives normalt i procent. Ækvivalenspartitionering kan anvendes på alle testniveauer.

4.2.2 Grænseværdianalyse

Grænseværdianalyse (GVA) er en udvidelse af ækvivalenspartitionering, der udelukkende kan anvendes, når partitionerne er sorteret, så de består af numeriske eller sekventielle data. En partitions mindste og største værdier (eller første og sidste værdier) er dens grænseværdier (se Beizer 1990).

Eksempel: Hvis et input datafelt kun må acceptere heltalsværdier som input, kan man anvende et numerisk tastatur til at gøre indtastning af andre værdier end heltal umuligt. Hvis f.eks. det gyldige interval er fra 1 til 5,

så vil der være tre ækvivalenspartitioner: ugyldig (for lav); gyldig; ugyldig (for høj). For gyldige ækvivalenspartitioner er grænseværdierne 1 og 5. For ugyldige (for høje) partitioner er grænseværdien 6. For ugyldige (for lave) partitioner er der kun én grænseværdi, nemlig 0, fordi det er en partition med kun et medlem.

I eksemplet ovenfor identificerer vi to grænseværdier per grænse. Grænsen mellem ugyldig (for lav) og gyldig giver testværdierne 0 og 1. Grænsen mellem gyldig og ugyldig (for høj) giver testværdierne 5 og 6. Nogle variationer af denne teknik identificerer tre grænseværdier per grænse: værdierne inden, ved og lige over grænsen. I det tidligere eksempel, og ved at anvende trepunkts grænseværdier, er de nedre grænseværdier 0, 1 og 2 og de øvre grænseværdier 4, 5 og 6 (se Jorgensen 2014).

Det er mere sandsynligt, at der optræder ukorrekt adfærd ved ækvivalenspartitionens grænser end inde i partitionen. Det er vigtigt at huske, at både de specificerede og implementerede grænser kan være blevet forskudt til positioner over eller under deres tiltænkte positioner, de kan helt mangle eller de kan være blevet implementeret med uønskede ekstra grænser. Grænseværdianalyse og -test kan afsløre næsten alle defekter af den type ved at forsøge at få softwaren til at vise adfærd fra en anden partition end den, grænseværdien tilhører.

Grænseværdianalyse kan anvendes på alle testniveauer. Den kan generelt anvendes til at teste krav, der kræver en række tal som input (f.eks. datoer og tidspunkter). Grænseværdidækning for en partition måles som antallet af grænseværdier, der testes, divideret med det totale antal af identificerede grænsetestværdier. Den udtrykkes normalt i procent.

4.2.3 Beslutningstabeltest

Beslutningstabellen er en god måde at registrere komplekse forretningsregler som et system skal udføre. Testeren opstiller en beslutningstabel ved at identificere betingelser (ofte input) og systemets efterfølgende handlingerne (ofte output). Disse danner rækker i tabellen, typisk med betingelserne i toppen og handlingerne i bunden. Hver kolonne svarer til en beslutningsregel, der definerer en unik kombination af betingelser, som udløser den handling, som er forbundet med reglen. Betingelsernes og handlingernes værdier vises generelt som boolske værdier (sandt eller falsk) eller diskrete værdier (f.eks. rød, grøn, blå), men kan også være tal eller en talrække. Forskellige typer af betingelser og handlinger kan optræde i samme tabel.

Almindelige notation for beslutningstabeller:

For betingelser:

- J betyder, at betingelsen er sand (kan også vises som S eller 1)
- N betyder, at betingelsen er falsk (kan også vises som F eller 0)
- - betyder, at betingelsens værdi ikke er relevant (kan også vises som N/A)

For handlinger:

- X betyder, at handlingen bør finde sted (kan også vises som J eller S eller 1)
- Blank betyder, at handlingen ikke bør finde sted (kan også vises som – eller N eller F eller 0)

En hel beslutningstabel skal have tilstrækkeligt med kolonner (testcases) til at dække alle kombinationer af betingelser. Ved at udelade kolonner der ikke påvirker resultatet, kan antallet af testcases reduceres betydeligt. For eksempel ved at udelade umulige kombinationer af betingelser. For flere oplysninger om, hvordan man udelader kolonner, se ISTQB® CTAL-TA Certified Tester Advanced Level Test Analyst syllabus.

Den normale standard for minimal dækning for en beslutningstabeltest er almindeligvis at have mindst én testcase pr. beslutningsregel i tabellen. Det vil normalt betyde, at alle kombinationer af beslutninger er dækket.

Dækningen måles som et antal af beslutningsregler, der testes af mindst én testcase, divideret med det totale antal beslutningsregler. Den udtrykkes normalt i procent.

Beslutningstabeltestens styrke er, at den hjælper med at identificere alle de vigtige kombinationer af betingelser, hvoraf nogle af disse ellers ville kunne blive overset. Den hjælper også med at finde mangler i kravene. Den kan anvendes på ethvert testniveau i alle situationer, hvor softwarens adfærd afhænger af kombinationer af betingelser.

4.2.4 Tilstandsovergangstest

Komponenter eller systemer kan reagere forskelligt på en hændelse, afhængigt af den nuværende tilstand eller tidligere historie (f.eks. hændelsen, der har fundet sted, siden systemet blev igangsat). Den tidligere historie kan opsummeres ved at anvende konceptet med tilstande. Et tilstandsovergangsdiagram viser de mulige softwaretilstande, såvel som hvordan softwaren indgår i, udgår fra eller overgår mellem tilstande. En overgang igangsættes af en hændelse (f.eks. en brugers indtastning af en værdi i et felt). Hændelsen leder til en overgang. Samme hændelse kan lede til to eller flere forskellige overgange fra samme tilstand. Tilstandsovergangen kan lede til, at softwaren udfører en handling (f.eks. resultatet af en beregning eller fejlmeddelelse).

En tilstandsovergangstabel viser alle de gyldige overgange og potentielt ugyldige overgange mellem tilstande, såvel som hændelser og de efterfølgende handlinger med gyldige overgange. Tilstandsovergangsdiagrammer viser normalt kun de gyldige overgange og udelader de ugyldige overgange.

En test kan designes til at dække typiske tilstandssekvenser, at afvikle alle tilstande, at afvikle alle overgange, at afvikle specifikke overgangssekvenser eller at teste ugyldige overgange.

Tilstandsovergangstest anvendes til menubaserede applikationer og anvendes meget for indlejret software. Teknikkerne passer også til at bygge en model over et forretningsscenarie med særlige tilstande eller til at teste skærmnavigation. En tilstands koncept er abstrakt – den kan repræsentere et par linjer kode eller hele forretningsprocessen.

Dækningen bliver generelt målt på antallet af identificerede tilstande eller testede overgange, divideret med det totale antal af identificerede tilstande eller overgange i testobjektet. Dækningen udtrykkes normalt i procent. For flere oplysninger om dækningskriterierne, se ISTQB® CTAL-TA Certified Tester Advanced Level Test Analyst syllabus.

4.2.5 Usecasetest

Test kan udledes af usecases, hvor man med en specifik måde at designe interaktioner med software-elementer der indarbejder krav til softwarefunktionerne. Usecases forbindes med aktører (menneskelige brugere, ekstern hardware eller andre komponenter eller systemer) og subjekter (komponenten eller systemet, som usecasen anvendes på).

Hver usecase specificerer en adfærd, som et subjekt kan udføre i samarbejde med en eller flere aktører (UML 2.5.1 2017). En usecase kan beskrives ved sine interaktioner eller aktiviteter, såvel som ved sine start-betingelser, slutbetingelser og sit naturlige sprog, hvor det er passende. Interaktion mellem aktører og subjekt kan lede til ændringer i subjektets tilstand. Interaktioner kan repræsenteres grafisk af arbejdsstrømme, aktivitetsdiagrammer eller forretningsprocesmodeller.

En usecase kan indeholde forskellige variationer af sin basisadfærd, herunder usædvanlig adfærd og fejlhåndtering (systemets reaktion og gendannelse fra programmering, applikation og kommunikationsfejl, der f.eks. leder til fejlmeddelelser). Testen skal afvikle den definerede adfærd (basis, usædvanlig eller alternativ og fejlhåndtering). Dækningen kan måles på den andel af usecaseadfærd, der testes individuelt af den totale mængde af usecaseadfærd. Den udtrykkes normalt i procent.

For flere oplysninger om dækningskriterierne for usecase test, se ISTQB® CTAL-TA Certified Tester Advanced Level Test Analyst syllabus.

4.3 White-box testteknikker

White-box test er baseret på testobjektets interne strukturer. White-box teknikker kan anvendes på alle testniveauer, men de to koderelaterede teknikker, der beskrives i dette afsnit, er almindeligvis anvendte på komponenttestniveau. Der findes flere avancerede testteknikker, der anvendes i sikkerhedskritiske og missionskritiske miljøer og i miljøer med krav om høj integritet for at opnå en grundig dækning, men de beskrives ikke her. For flere oplysninger om sådanne teknikker, se ISTQB® CTAL-TTA Certified Tester Advanced Level Technical Test Analyst syllabus.

4.3.1 Instruktionstest og -dækning

Instruktionstest afvikler de mulige eksekverbare instruktioner i koden. Dækningen måles på det antal af instruktioner, der afvikles ved test, divideret af det totale antal af eksekverbare instruktioner i testobjektet. Den udtrykkes normalt i procent.

4.3.2 Beslutningstest og -dækning

Beslutningstest afvikler beslutninger i koden og tester, om den afviklede kode er baseret på beslutnings-resultatet. For at kunne gøre det, følger testcases et kontrolflow, der opstår ved et beslutningspunkt (f.eks. for en IF-instruktion, en for det sande resultat og en for det falske resultat; for en CASE-instruktion kræves der testcases for alle de mulige resultater, herunder standardresultatet).

Dækningen måles på antal af beslutningsresultater, der afvikles af testen, divideret med det samlede antal beslutningsresultater i testobjektet. Den udtrykkes normalt i procent.

4.3.3 Værdien af instruktions- og beslutningstest

Når man har opnået 100% instruktionsdækning, sikrer det, at alle eksekverbare instruktioner i koden er blevet testet mindst én gang, men det sikrer ikke, at beslutningslogikken er testet. Ud af de to white-box teknikker beskrevet i denne syllabus giver beslutningstest samme eller højere dækning end instruktionstest

Når man har opnået 100% beslutningsdækning, afvikler testen alle beslutningsresultater, hvilket giver test af samtlige sande og falske resultater, selv når der ikke findes nogen eksplicitte falske instruktioner (f.eks. i tilfældet af en IF-instruktion uden ELSE i koden). Instruktionsdækning hjælper med at finde defekter i koden, som ikke afvikles af andre tests. Beslutningsdækning hjælper med at finde defekter i koden, hvor andre tests ikke har både de sande og falske resultater.

100% beslutningsdækning garanterer 100% instruktionsdækning (mens det omvendte ikke er tilfældet).

4.4 Erfaringsbaserede testteknikker

Når man anvender erfaringsbaserede testteknikker, udledes testcasene af testerens færdigheder og intuition og af testerens erfaringer med lignende applikationer og teknologier. Disse teknikker kan være nyttige til at udarbejde tests, der ikke er nemme at identificere med mere systematiske teknikker. Afhængigt af testerens tilgang og erfaring kan sådanne teknikker give en bred vifte af tests med varierende grader af dækning og effektivitet. Dækningen kan være svær at afgøre og kan være umulig at måle for disse teknikker.

Alment anvendte erfaringsbaserede teknikker beskrives i de følgende afsnit.

4.4.1 Fejlgætning

Fejlgætning er en teknik, der anvendes til at forudsige hvor fejl, defekter og afvigelser finder sted. Den er baseret på testerens viden, herunder:

- Hvordan applikationen tidligere har fungeret
- Hvilke typer fejl ofte begået
- Fejl, der er fundet i andre applikationer

En metodisk tilgang til fejlgætningsteknikken er at lave en liste over mulige fejl, defekter og afvigelser, og designe tests, der kan afsløre afvigelser og de defekter, som har forårsaget dem. Disse lister over fejl, defekter og afvigelser kan baseres på erfaringer, data over defekter og afvigelser og fra fælles viden om, hvorfor softwaren fejler.

4.4.2 Udforskende test

I udforskende test designes, afvikles, logges og evalueres uformelle (ikke forud definerede) tests dynamisk, mens testen afvikles. Testens resultater anvendes til at lære mere om komponenten eller systemet og til at skabe tests til områder, der kræver mere test.

Udforskende test udføres nogle gange som sessionsbaseret test for at strukturere aktiviteten. I sessions-baserede test udføres en udforskende test inden for en foruddefineret tidsramme. Testeren anvender et test-charter, der indeholder testformål som ledetråde i test. Testeren kan anvende testsessionsark til at dokumentere de trin, der følges, og de fund, der gøres.

Udforskende test er mest anvendelig, når der er få eller utilstrækkelige specifikationer eller betydeligt tidspres på testen. Udforskende test er også anvendelig som supplement til mere formelle testteknikker.

Udforskende test er stærkt forbundet med reaktive teststrategier (se afsnit 5.2.2). Udforskende test kan benytte black-box, white-box og erfaringsbaserede teknikker.

4.4.3 Tjeklistebaserede test

I tjeklistebaseret test designer, implementerer og afvikler testerne test til at dække testbetingelserne, der kan findes i tjeklisten. Som en del af analysen laver testerne nye tjeklister eller udvider eksisterende tjeklister, men testerne kan også anvende eksisterende tjeklister uden ændringer. Sådanne tjeklister kan opbygges på baggrund af erfaring, viden om, hvad der er vigtigt for brugeren, eller en forståelse af, hvorfor og hvordan software kan fejle.

Tjeklister kan laves til at understøtte forskellige testtyper, herunder funktionelle og ikke-funktionelle tests. Hvis der ikke findes detaljerede testcases, kan tjeklistebaseret test give retningslinjer og en grad af sammenhæng. Da det drejer sig om lister på højt niveau, er det meget sandsynligt, at variation i de faktiske test kan finde sted, hvilket leder til potentiale for bedre dækning, men mindre repeterbarhed.

5. Teststyring — 225 minutter

Nøgleord

defekthåndtering, defektrapport, konfigurationsstyring, produktrisiko, projektrisiko, risiko, risikobaseret test, risikoniveau, slutkriterier, startkriterier, tester, testestimering, testfremdriftsrapport, testkontrol, testmanager, testopsummeringsrapport, testovervågning, testplan, testplanlægning, teststrategi, testtilgang

Læringsmål for teststyring

5.1 Testorganisation

FL-5.1.1 (K2) Forklar fordele og ulemper ved uafhængig test
FL-5.1.2 (K1) Identificer en testmanagers og en testers opgaver

5.2 Testplanlægning og -estimering

FL-5.2.1 (K2) Opsummer en testplans formål og indhold
FL-5.2.2 (K2) Forklar forskellene mellem forskellige teststrategier
FL-5.2.3 (K2) Giv eksempler på potentielle start- og slutkriterier
FL-5.2.4 (K3) Anvend viden om prioritering og tekniske og logiske afhængigheder for at planlægge en testafvikling for et antal testcases
FL-5.2.5 (K1) Identificer de faktorer, der påvirker den indsats, der har med testen at gøre
FL-5.2.6 (K2) Forklar forskellen mellem to estimeringsteknikker: den metrikbaserede teknik og den ekspertbaserede teknik

5.3 Testovervågning og -kontrol

FL-5.3.1 (K1) Husk metrikker, der anvendes til test
FL-5.3.2 (K2) Opsummer testrapportens formål, indhold og målgruppe

5.4 Konfigurationsstyring

FL-5.4.1 (K2) Opsummer, hvordan konfigurationsstyring understøtter test

5.5 Risiko og test

FL-5.5.1 (K1) Definer risikoniveauet ved at anvende sandsynlighed og effekt
FL-5.5.2 (K2) Forklar forskellen mellem projekt- og produktrisici
FL-5.5.3 (K2) Beskriv ved hjælp af eksempler, hvordan produktrisikoanalysen kan påvirke testens grundighed og omfang

5.6 Defekthåndtering

FL-5.6.1 (K3) Skriv en defektrapport ved at gennemgå de defekter, der er fundet i test

5.1 Testorganisation

5.1.1 Uafhængig test

Testopgaver kan udføres af personer i særlige testroller eller af personer i andre roller (f.eks. kunder). En særlig grad af uafhængighed gør ofte testeren mere effektiv til at finde defekter pga. forskellene mellem forfatterens og testerens kognitive holdninger (se afsnit 1.5). Uafhængighed er derimod ikke en erstatning for kendskab, og udviklerne kan finde defekter i deres egen kode på effektiv vis.

Grader af uafhængighed i test omfatter ikke (fra et lavt niveau af uafhængighed til et højt niveau):

- Ingen uafhængige testere; den eneste form for uafhængig test er, at udviklerne selv tester deres egen kode
- Uafhængige udviklere eller testere inden for udvikler- eller projektteamet; det kan være udviklere, der tester deres kollegers produkter
- Uafhængige testteams eller -grupper inden for organisationen, der rapporterer til projektledelsen eller topledelsen
- Uafhængige testere fra organisationen eller brugermiljøet eller testere specialiseret i særlige testtyper så som brugervenlighed, sikkerhed, performance, lovgivning/overensstemmelse eller flytbarhed
- Uafhængige testere uden for organisationen, der enten arbejder på stedet (in-house) eller eksternt (outsource)

For de fleste typer af projekter er det normalt bedst at have flere testniveauer, hvoraf nogle af disse udføres af uafhængige testere. Udviklerne bør deltage i testen, særligt på lavere niveauer, for at være med til at kontrollere kvaliteten af deres eget arbejde.

Måden, som testens uafhængighed implementeres på, varierer alt efter softwareudviklingens livscyklusmodel. F.eks. kan testere være en del af et udviklingsteam i agil udvikling. I nogle organisationer, der anvender den agile metode, kan disse testere på samme tid betragtes som en del af det større team af testere. Derudover kan product owners i sådanne organisationer udføre accepttest for at validere userstories ved slutningen af hver iteration.

Fordelene ved testuafhængighed omfatter:

- Uafhængige testere finder andre typer af afvigelser sammenlignet med udviklere pga. deres anderledes baggrund, tekniske perspektiver og biaser
- En uafhængig tester kan verificere, udfordre og modbevise interessenternes antagelser mens der foregår specifikation og implementering af systemet
- Uafhængige testere fra en leverandør kan rapportere på en sober og objektiv måde om det system der er under test uden (politisk) pres fra den forretning som har hyret dem

Mulige ulemper ved testuafhængighed omfatter:

- Isolation fra udviklingsteamet, der kan lede til manglende samarbejde, forsinkelser i feedback til udviklingsteamet eller et fjendtligt forhold til udviklingsteamet
- Udviklerne kan føle, at de mister deres ansvaret for kvaliteten
- Uafhængige testere kan ses som en flaskehals
- Uafhængige testere kan mangle vigtige oplysninger (f.eks. om testobjektet)

Mange organisationer er i stand til at opnå fordelene ved testuafhængighed, mens de samtidig undgår ulemperne.

5.1.2 En testmanagerens og testerens opgaver

I denne syllabus dækkes to testroller: testmanager og tester. De aktiviteter og opgaver, der udføres af de to roller, afhænger af projektets og produktets kontekst, evnerne for personerne i de forskellige roller og af selve organisationen.

Testmanageren har det overordnede ansvar for en vellykket ledelse af testaktiviteterne. Testmanagerens rolle kan udføres af en professionel testmanager eller af en projektleder, udviklingsleder eller en kvalitetssikringsleder. I større projekter eller organisationer kan flere testteams rapportere til testmanageren, testcoachen eller testkoordinatoren, og hvert team ledes af en testleder eller ledende tester.

Typiske opgaver for en testmanager:

- Udvikling eller review af testpolitik og teststrategi for organisationen
- At planlægge testaktiviteter ved at overveje konteksten og forstå testformålet og -risici. Dette kan indebære testtilgange at estimere testtiden, -indsatsen og -omkostningerne, der definerer testniveauerne og testcyklusserne, og planlægning af defekthåndtering
- At skrive og opdatere testplaner
- At koordinere testplanen med projektlederen, product owners og evt. andre parter
- At påvirke projektplanlægningen med testperspektiver, f.eks. ved planlægning af integration
- At igangsætte analyse, design, implementering og afvikling af tests, at overvåge testens fremdrift og resultater, og at tjekke slutkriteriernes status (eller Definition of Done), og faciliterer testafslutningsaktiviteter
- At forberede og levere testfremdriftsrapporter og testopsummeringsrapporter, baseret på de indsamlede oplysninger
- At tilpasse planlægningen baseret på testresultater og -fremdrift (nogle gange dokumenteret i testfremdriftsrapporter og/eller i testopsummeringsrapporter for andre tests, der allerede er færdiggjort i projektet) og foretag alle handlinger, der er nødvendige for testkontrol
- At understøtte opsætning af et defektstyringssystem og passende konfigurationsstyring af testware
- At introducere passende metrikker for måling af testfremdrift og evaluere testens og produktets kvalitet
- At understøtte udvælgelse og implementering af værktøj for at støtte testprocessen, herunder ved at anbefale budget for værktøjsvalg (og muligvis foretage indkøb og/eller understøtte indkøbet), tildele tid og indsats til pilotprojekter, og yde vedvarende support til anvendelsen af værktøjerne.
- At foretage beslutninger vedrørende implementering af testmiljøer
- At fremme og være fortaler for testere, testteamet og testprofessionen inden for organisationen
- At udvikle testernes færdigheder og karriere (f.eks. ved uddannelsesplaner, performanceevaluering, coaching osv.)

Den måde, testmanagerens rolle skal udføres på, afhænger af softwareudviklingens livscyklus. F.eks. udføres de daglige ledelsesopgaver i agil udvikling ofte af en tester, der arbejder i teamet. Opgaver, der spænder over adskillige teams eller hele organisationen, eller som har med personalestyring at gøre, kan udføres af testmanagere uden for udviklingsteamet (testcoaches). Se Black 2009 for flere oplysninger om håndtering af testprocessen.

Typiske opgaver for en tester:

- Review af og bidrag til testplaner
- Analyse, review og vurdering af krav, userstories og acceptkriterier, specifikationer og modeller for testbarhed (f.eks. testgrundlag)
- Identifikation og dokumentation af testbetingelser. Etablering af sporbarhed mellem testcases, testbetingelser og testgrundlag

- Design, opsætning af og kontrol af testmiljøer, ofte koordineret med systemadministration og netværksstyring
- Design og implementering af testcases og testprocedurer
- Forberedelse og indsamling af testdata
- Udarbejdelse af detaljerede testafviklingsplaner
- Afvikling af test, evaluering af resultater og dokumentation af afvigelser fra forventede resultater
- Anvendelse af passende værktøjer til at facilitere testprocesserne
- Automatisering af test efter behov (kan understøttes af en udvikler eller testautomatiseringsekspert)
- Evaluering af ikke-funktionelle egenskaber som f.eks. effektivitet, pålidelighed, brugervenlighed, sikkerhed, kompatibilitet og flytbarhed
- Review af tests, der er udviklet af andre

Personer, der arbejder med testanalyser, testdesign, særlige testtyper eller testautomatisering, kan være specialister i disse roller. Afhængigt af de risici, der er forbundet med produktet og projektet, og den valgte livscyklusmodel for softwareudviklingen, kan forskellige personer overtage testerrollen på forskellige testniveauer. F.eks. vil testerens rolle på komponenttestniveau og komponentintegrationsniveau ofte blive håndteret af udviklere. På accepttestniveau udføres testerrollen ofte af en forretningsanalytiker, fageksperter og brugere. På systemtestniveau og systemintegrationstestniveau udføres testerrollen ofte af et uafhængigt testteam. På driftsmæssigt accepttestniveau udføres testerrollen ofte af personale fra drift eller system-administration.

5.2 Testplanlægning og -estimering

5.2.1 Testplanens formål og indhold

En testplan skitserer testaktiviteterne for udviklings- og vedligeholdelsesprojekter. Planlægningen er påvirket af organisationens testpolitik og teststrategi, de anvendte udviklingslivscyklusser og -metoder (se afsnit 2.1), testens omfang, formål, risici, rammer, kritikalitet, testbarhed og resursernes tilgængelighed.

Testplanen er en kontinuerlig aktivitet og udføres gennem produktets livscyklus. I løbet af et projekt bliver flere oplysninger tilgængelige, og man kan tilføje flere detaljer i testplanen (bemærk, at produktets livscyklus kan strække sig ud over projektets omfang for at inkludere en vedligeholdelsesfase). Feedback fra testaktiviteter bør anvendes til at opdage nye risici, så planlægningen kan justeres. Planlægningen kan dokumenteres i en hovedtestplan og i separate testplaner for testniveauerne, så som systemtest og accepttest eller for separate testtyper, så som brugervenlighedstest og performancetest. Testplanlægningsaktiviteter kan indeholde det følgende, og nogle af disse kan dokumenteres i en testplan:

- Omfang, formål og testrisici
- Definition af den overordnede testtilgang
- Integration og koordinering af testaktiviteter i softwarelivscyklusaktiviteterne
- Beslutning om, hvad der skal testes, de personer og andre resurser, der er behov for at udføre forskellige testaktiviteter, og hvordan testaktiviteterne skal udføres
- Planlægning af testanalyse, -design, -implementering, -afvikling og evalueringsaktiviteter på særlige datoer (f.eks. i sekventiel udvikling) eller ved hver iteration (f.eks. i iterativ udvikling)
- Valg af metrikker for testovervågning og -kontrol
- Budgettering for testaktiviteter
- Fastlæggelse af detaljeniveau og testdokumentationens struktur (f.eks. med skabeloner eller dokumenter med eksempler)

Testplanernes indhold varierer og kan strække sig ud over de ovennævnte emner. Et eksempel på en testplanstruktur og en testplan kan findes i ISO-standard (ISO/IEC/IEEE 29119-3).

5.2.2 Teststrategi og testtilgang

En teststrategi giver en generaliseret beskrivelse af testprocessen, normalt på produkt- eller organisationsniveau. Almindelige typer teststrategier omfatter:

- **Den analytiske:** Denne type teststrategi er baseret på en analyse af en faktor (f.eks. krav eller risici). Risikobaseret test er et eksempel på en analytisk tilgang, hvor testen er designet til og prioriteret på grundlag af risikoniveauet
- **Den modelbaserede:** I denne type teststrategi er testene designet på baggrund af en model over kravene til produktet, så som funktion, en forretningsproces, en intern struktur eller en ikke-funktionel egenskab (f.eks. pålidelighed). Eksempler: forretningsproceduremodeller, tilstands-modeller og pålidelighedsvækstmodeller
- **Den metodiske:** Denne type teststrategi afhænger af systematisk brug af prædefinerede sæt af tests eller testbetingelser, så som en taksonomi over almindelige eller typiske afvigelser, en liste over vigtige kvalitetsegenskaber eller mobilapps eller hjemmesiders oplevede standarder på tværs af firmaet
- **Procesoverensstemmende** (eller standardoverensstemmende)**:** Denne type teststrategi indebærer analyse, design og implementering af tests, på grundlag af eksterne regler og standarder som f.eks. industrispecifikke standarder ved procesdokumentation, ved grundig identifikation og brug af testgrundlag eller ved en anden proces eller standard påført på eller af organisationen
- **Den instruerede** (eller rådgivende)**:** Denne type teststrategi er primært drevet af rådgivning, vejledning eller interessenternes instruktioner, eksperter inden for fagområder eller teknologieksperters råd, vejledning eller instruktioner. Rådgiverne kan være uden for testteamet eller uden for selve organisationen
- **Regressionsmodvirkning:** Denne type teststrategi er motiveret af et ønske om at undgå regression af eksisterende funktioner. Denne teststrategi medfører genbrug af eksisterende testware (særligt testcases og testdata), omfattende automatisering af regressionstests og standard testsuiter
- **Reaktiv:** I denne type teststrategi reagerer testen på den komponent eller det system, der bliver testet, og hændelser finder sted under testens afviklingen, snarere end at være planlagt på forhånd som de foregående strategier. Test bliver designet, implementeret og afviklet på grundlag af viden fra tidligere testresultater. Det er almindeligt at bruge udforskende test ved reaktiv strategi

En passende teststrategi skabes ofte ved at kombinere flere af disse typer teststrategier. F.eks. kan risikobaseret test (en analytisk strategi) kombineres med udforskende test (en reaktiv strategi); de komplementerer hinanden og man opnår en mere effektiv test, når de anvendes sammen.

Mens teststrategien giver en generaliseret beskrivelse af testprocessen, tilpasser testtilgangen teststrategien til et særligt projekt eller udgivelse. Testtilgangen er udgangspunktet for valget af testteknikker, testniveauer og testtyper, og definerer startkriterier og slutkriterier (eller henholdsvis Definitionen of Ready og Definition of Done). Tilpasningen af strategien baseres på de beslutninger, der er foretaget i relation til projektets kompleksitet og mål, det udviklede produkts type og produktrisikoanalyse. Den valgte tilgang afhænger af konteksten og kan betragtes som risici, sikkerhed, tilgængelige resurser og færdigheder, teknologi, systemets natur (f.eks. specialbygget kontra COTS), testformål og regulativer.

5.2.3 Startkriterier og slutkriterier (Definition of Ready og Definition of Done)

For at udøve effektiv kontrol over softwarens kvalitet og af selve testen, anbefales det at have kriterier, der definerer, hvornår en givet testaktivitet bør igangsættes, og hvornår aktiviteten er færdig. Startkriterier (der typisk kaldes Definition of Ready i agil udvikling) definerer startbetingelserne for at foretage en givet testaktivitet. Hvis startkriterierne ikke er opfyldt, er det sandsynligt, at aktiviteten viser sig at være sværere, mere tidskrævende, mere omkostningsfuld og mere risikabel at teste. Slutkriterierne (der i agil udvikling typisk kaldes for Definition of Done) definerer, hvilke betingelser, der skal opnås for erklære, at et testniveau eller en række af tests er blevet udført. Start- og slutkriterier bør defineres for hvert testniveau og testtype og er forskellige afhængigt af testformålet

Typiske startkriterier omfatter:

- Tilgængelighed af testbare krav, userstories og/eller modeller (f.eks. ved modelbaseret teststrategi)
- Tilgængelighed af testelementer, der lever op til slutkriterierne for ethvert tidligere testniveau
- Tilgængelighed af testmiljøer
- Tilgængelighed af de nødvendige testværktøjer
- Tilgængelighed af testdata og andre nødvendige resurser

Typiske slutkriterier omfatter:

- De planlagte tests er udført
- Et forud defineret niveau af dækning (f.eks. krav, userstories, acceptkriterier, risici, kode) er opnået
- Antallet af uløste defekter under den aftalte grænse
- Antallet af skønnede, tilbageværende defekter er tilstrækkeligt lavt
- De vurderede niveauer for pålidelighed, effektivitet, brugbarhed, sikkerhed og andre relevante kvalitetsegenskaber bedømmes som opfyldt

Det sker ofte at testaktiviteterne bliver stoppet uden at slutkriterierne er nået, hvis budgettet er overskredet, den planlagte tidsramme er overskredet og/eller der er behov for straks at sende produktet på markedet. Det kan være acceptabelt at afslutte testen under sådanne omstændigheder, hvis projektets interessenter og forretningsejere har reviewet og accepteret risikoen for at gå i produktion uden yderligere test.

5.2.4 Testafviklingsplanen

Når de forskellige testcases og testprocedurer produceres (potentielt med nogle testprocedurer automatiseret) og samles i testsuites, kan testsuites organiseres i en testafviklingsplan, der definerer den rækkefølge, de skal afvikles i. Testafviklingsplanen bør tage faktorer så som prioritering, afhængigheder, gentests, regressionstests og den mest effektive rækkefølge for afviklingen af testen i betragtning.

Ideelt set bør testcases arrangeres, så de afvikles med testcases med den højeste prioritet først. Dog kan det være tilfældet, at denne praksis ikke vil fungere, hvis disse testcases har afhængigheder, eller at funktionerne har afhængigheder. Hvis en testcase med en højere prioritet er afhængig af en testcase med en lavere prioritet, skal den afvikles først. Tilsvarende hvis der findes afhængigheder på tværs af testcases, skal de organiseres i en passende rækkefølge uanset deres relative prioriteter. Gen- og regressionstests skal også arrangeres efter prioritet på grundlag af behovet for hurtig feedback ved ændringer.

I nogle tilfælde er forskellige testsekvenser med forskellige effektivitetsniveauer mulige. I sådanne cases skal effektiviteten af testens afvikling efterstræbes, snarere end at følge prioriteringen.

5.2.5 Faktorer, der påvirker testindsatsen

Vurderingen af testindsatsen indebærer at skønne omfanget af det testarbejde, der er nødvendigt for at imødekomme målene med test af et projekt, en udgivelse eller en iteration. De faktorer, der påvirker testindsatsen, kan indeholde produktets karakteristika, udviklingsprocedurens karakteristika, personernes karakteristika og de testresultater, der vises nedenfor.

Produktets karakteristika

- De risici, der er knyttet til produktet
- Testgrundlagets kvalitet
- Produktets størrelse
- Produktdomænets kompleksitet

- Krav til kvalitetsegenskaber (f.eks. sikkerhed, pålidelighed)
- Testdokumentationens påkrævede detaljeniveau
- Krav om at gældende love og regler skal følges

Udviklingsprocessens karakteristika

- Organisationens stabilitet og modenhed
- Den anvendte udviklingsmodel
- Testtilgang
- De anvendte værktøjer
- Testprocessen
- Tidspres

Personernes karakteristika

- De involverede personers evner og erfaringer, særligt med lignende projekter og produkter (f.eks. domæneviden)
- Teamets samhørighed og ledelse

Testresultater

- Antallet og alvorligheden af de fundne defekter
- Det påkrævede omfang af genarbejde

5.2.6 Testestimeringsteknikker

Der findes et antal af estimeringsteknikker, der anvendes til at afgøre den påkrævede indsats for tilstrækkelig test. To af de mest almindeligt anvendte teknikker er:

- Den metrikbaserede teknik: vurderer testindsatsen på baggrund af lignende projekters metrikker eller baseret på typiske værdier
- Den ekspertbaserede teknik: vurderer testindsatsens på baggrund af testopgavernes ejeres erfaring eller af eksperter

F.eks. i agil udvikling er burndown charts eksempler på den metrikbaserede tilgang, hvor den tilbageværende indsats dokumenteres og rapporteres, og derefter bruges til at forbedre teamets arbejdshastighed og afgøre omfanget af det arbejde, som teamet kan udføre i næste iteration; hvorimod planningpoker er et eksempel på en ekspertbaseret tilgang, hvor hvert medlem af teamet vurderer indsatsen for at levere en funktion baseret på deres erfaring (ISTQB® CTFL-AT Certified Tester Foundation Level Agile Tester syllabus).

Inden for sekventielle projekter, er defektfjernelsesmodeller eksempler på den metrikbaserede tilgang, hvor volumen af defekter og tiden, det tager at fjerne dem optages og indrapporteres, hvilket giver grundlaget for en vurdering af fremtidige projekter af lignende art; hvorimod Wideband Delphi-estimeringsteknikken er et eksempel på en ekspertbaseret tilgang, hvori grupper af eksperter giver deres estimater, baseret på deres erfaringer (ISTQB® CTAL-TM Certified Tester Advanced Level Test Manager syllabus).

5.3 Testovervågning og -kontrol

Formålet med testovervågning er at indhente oplysninger og give feedback og synlighed om testaktiviteterne. Oplysningerne, der skal overvåges, kan indhentes manuelt eller automatisk og bør anvendes til at vurdere

testfremdrift og måle, om testens slutkriterier eller testopgaverne, der er tilknyttet et agil projekts definition of done, imødekommes, så som at nå målene for dækning af produktrisici, krav eller acceptkriterier.

Testkontrol beskriver alle vejledende eller korrigerende handlinger, der skal foretages som følge af de indsamlede og (muligvis) indrapporterede oplysninger og metrikker. Handlingerne kan dække enhver testaktivitet og kan påvirke alle andre softwarelivscyklusaktiviteter.

Eksempler på testkontrolhandlinger indebærer:

- Omprioritering af test, når en identificeret risiko finder sted (f.eks. at softwaren leveres for sent)
- Ændring af testplan pga. tilgængelighed eller utilgængelighed af testmiljø eller andre resurser
- Ny bedømmelse af om et testelement opfylder start- eller slutkriterier efter ændringer

5.3.1 Metrikker anvendt i test

Metrikker kan indsamles i løbet af og i slutningen af testaktiviteterne for at vurdere:

- Fremdriften sammenlignet med plan eller budget
- Testelementernes nuværende kvalitet
- Om testtilgangen er tilstrækkelig
- Testaktiviteternes effektivitet i henhold til målsætningerne

Almindelig testmetrik omfatter:

- Den procentdel af det planlagte arbejde der er udført ved forberedelse af testcases (eller procentdel af de planlagte testcases, der er implementeret)
- Procentdel af det planlagte arbejde, der er udført ved forberedelse af testmiljø
- Afvikling af testcases (f.eks. antal gennemførte og fejlede testcases, antal godkendte og afviste testcases og/eller antal godkendte og afviste testbetingelser)
- Defektoplysninger (f.eks. defekttæthed, defekter fundet og udbedret, afvigelsesrate og resultater fra gentest)
- Testdækning af krav, userstories, acceptkriterier, risici og kode
- Færdiggørelse af opgaver, resursetildeling og -anvendelse og indsats
- Testens omkostninger inkl. cost-benefit: Fordele ved at finde næste defekt sammenholdt med omkostningerne ved at fortsætte testen

5.3.2 Formål, indhold og testrapporternes målgruppe

Formålet med testrapportering er at opsummere og kommunikere oplysninger om testaktiviteten, både i løbet af og i slutningen af testaktiviteten (f.eks. et testniveau). Den testrapport, der udarbejdes i løbet af en testaktivitet kaldes en testfremdriftsrapport, hvorimod en testrapport, der udarbejdes i slutningen af en testaktivitet, kaldes en testopsummeringsrapport.

I løbet af testovervågningen og -kontrol udsender testmanageren testfremdriftsrapporter til interessenter. Ud over det almene indhold i testfremdriftsrapporterne og testopsummeringsrapporterne, kan de typisk også indeholde:

- Testaktiviteternes og -fremdriftens status i forhold til testplanen
- Faktorer, der hindrer fremdriften
- Planlagt test i næste rapporteringsperiode
- Testelementets kvalitet

Når slutkriterierne er nået, udsender testmanageren testopsummeringsrapporten. Denne rapport giver en opsummering af den udførte test, baseret på den seneste testfremdriftsrapport og andre relevante oplysninger.

Typiske testopsummeringsrapporter indeholder:

- Opsummering af den udførte test
- Oplysninger om, hvad der er sket i løbet af testperioden
- Afvigelser fra planen, herunder afvigelser i testaktiviteternes tidsplan, varighed og indsats
- Testens og produktkvalitetens status med hensyn til slutkriterier og definition of done
- Faktorer, der har blokeret eller fortsat blokerer fremdriften
- Metrik for defekter, testcases, testdækning, aktivitetsfremdrift og resurseforbrug (f.eks. som beskrevet i afsnit 5.3.1)
- Resterende risici (se afsnit 5.5)
- De genbrugelige testarbejdsprodukter, der er produceret

Indholdet i en testrapport kan afhænge af projektet, de organisatoriske krav og softwareudviklingens livscyklus. F.eks. kan et komplekst projekt med mange interessenter eller et lovreguleret projekt kræve flere detaljer og mere grundig rapportering end en hurtig softwareopdatering. En testfremdriftrapport i et agilt projekt kan indarbejdes i taskboards, defektopsummeringer og burndown charts, der kan diskuteres ved det daglige standupmøde (se ISTQB® Foundation Level Agile Tester Extension syllabus).

Ud over tilpasse testrapporterne til projektets kontekst, bør testrapporterne også tilpasses rapporternes målgruppe. Typen og antal af oplysninger, til en teknisk målgruppe eller testteam er forskellig fra dét der skal med i et management summary. I det første tilfælde kan detaljerede oplysninger om defekttyper og tendenser være vigtigt. I det andet tilfælde kan en rapport på højt niveau (f.eks. en statusopsummering over defekter efter prioritering, budget, tidsplan og testbetingelser bestået/fejlet/ikke testet) være mere passende.

ISO-standard (ISO/IEC/IEEE 29119-3) henviser til to typer af testrapporter, testfremdriftsrapporter og testafslutningsrapporter (der kaldes testopsummeringsrapporter i denne syllabus) og indeholder strukturer og eksempler for hver type.

5.4 Konfigurationsstyring

Formålet med konfigurationsstyring er at etablere og vedligeholde komponentens eller systemets integritet, såvel som testwaren og deres indbyrdes forhold gennem projektet og produktets livscyklus.

For at kunne understøtte testen korrekt, kan konfigurationsstyringen omfatte:

- At alle testelementer er identificeret unikt, versionen kontrolleret, og at ændringer spores og relateres til hinanden
- At alle af testwarens testelementer er identificeret unikt, versionstyret, og at ændringer spores, relateres til hinanden og relateres til versioner af testelementer, så sporbarheden kan opretholdes gennem hele testprocessen
- At der refereres utvetydigt til alle identificerede dokumenter og softwareelementer i testdokumentationen

Under testplanlægningen bør konfigurationsstyringsprocedurer og infrastruktur (værktøjer) identificeres og implementeres.

5.5 Risici og test

5.5.1 Definition af risiko

En risiko indebærer muligheden for en fremtidig hændelse med negative konsekvenser. Risikoniveauet afgøres ud fra sandsynligheden for hændelsens opståen og effekten (skaden) fra hændelsen.

5.5.2 Produkt- og projektrisici

Produktrisiko indebærer muligheden for, at et arbejdsprodukt (f.eks. en specifikation, komponent eller test) ikke lever op til brugernes og/eller interessenternes forventede behov. Produktrisici afhænger af et produkts specifikke kvalitetsegenskaber, f.eks. funktionel egnethed, pålidelighed, præstationens effektivitet, brugbarhed, sikkerhed, kompatibilitet, vedligeholdelsesevne og portabilitet. Produktrisici kaldes også kvalitetsrisici. Eksempler på produktrisici:

- Softwaren er muligvis ikke i stand til at udføre sin tiltænkte funktion i henhold til specifikationerne
- Softwaren er muligvis ikke i stand til at udføre sin tiltænkte funktion i henhold til brugerens, kundens og/eller interessentens behov
- Systemarkitekturen understøtter muligvis ikke de ikke-funktionelle krav på passende måde
- Specifikke beregninger bliver i nogle tilfælde udført ukorrekt
- Et loopkontrolflow kan være kodet ukorrekt
- Svartider kan være utilstrækkelige for et højtydende transaktionsbehandlende system
- User experience (UX) lever muligvis ikke op til forventningerne

Projektrisiko medfører situationer, der - hvis de finder sted - kan have en negativ effekt på projektets evne til at leve op til sine mål. Eksempler på projektrisici omfatter:

- Problemer for projektet:
  - Forsinkelser ved levering, opgavens afslutning eller definition of done
  - Ukorrekte vurderinger, omplacering af resurser til projekter med højere prioritet eller generelle besparelser på tværs af organisationen kan betyde utilstrækkelige resurser til projektet
  - Sene ændringer kan lede til omfattende omarbejde
- Organisatoriske problemer:
  - Utilstrækkelige færdigheder, træning og personale
  - Personaleproblemer kan forårsage konflikter og problemer
  - Brugere, forretningen eller fageksperter kan være utilgængelige pga. modstridende forretningsprioriteter
- Politiske spørgsmål:
  - Testere kan ikke kommunikere deres behov og/eller testresultaterne tilstrækkeligt
  - I tilfældet af, at udviklere og/eller testere ikke formår at følge op på de oplysninger, der er fundet i test og reviews (f.eks. ikke at forbedre udviklings- og testpraksisser)
  - Der kan være en upassende attitude eller forventninger til test (f.eks. ikke at værdsætte værdien af at finde defekter i løbet af test)
- Tekniske problemer:
  - Kravene er ikke defineret godt nok
  - Kravene kan ikke imødekommes inden for de eksisterende rammer
  - Testmiljøerne er muligvis ikke klar i tide
  - Datakonvertering, migrationsplanlægning og deres værktøjssupport kan være forsinket
  - Svagheder i udviklingsprocessen kan påvirke projektets arbejdsprodukts konsistens eller kvalitet, så som i design, kode, konfiguration, testdata og testcases
  - Ringe defektstyring og lignende problemer kan resultere i akkumulerede defekter eller anden teknisk gæld
- Leverandørproblemer:

- En tredjepart lever muligvis ikke op til leverancen af et nødvendigt produkt eller service, eller er gået konkurs
- Kontraktlige problemer kan forårsage problemer for projektet

Projektrisici kan både påvirke udviklingsaktiviteter og testaktiviteter. I nogle tilfælde er projektledere ansvarlige for at håndtere projektrisici, men det er ikke usædvanligt for testmanagere at have ansvaret for testrelaterede projektrisici.

5.5.3 Risikobaseret test og produktkvalitet

Den vurderede risiko anvendes til at fokusere indsatsen i løbet af testen. Den anvendes til at afgøre, hvor og hvornår testen skal igangsættes og til at identificere områder, der behøver mere opmærksomhed. Test anvendes til at reducere sandsynligheden for, at negative hændelser finder sted eller for at reducere de negative effekter. Test anvendes som risikoreducerende aktiviteter til at yde information om identificerede risici, såvel som at yde information om resterende (uafklarede) risici.

En risikobaseret tilgang til test giver proaktive muligheder for at reducere produktets risikoniveauer. Det indebærer produktrisikoanalyse og identifikation af produktrisici og vurdering af hver risikos sandsynlighed og effekt. De efterfølgende oplysninger om produktrisici anvendes til at vejlede testplanlægning, specificering, forberedelse og afvikling af testcases og testovervågning og -styring. Tidlig analyse af produktrisici bidrager til et vellykket projekt.

I den risikobaserede tilgang anvendes produktrisikoanalysen til:

- At bestemme de testteknikker, der skal anvendes
- At bestemme de specifikke testniveauer og -typer, der skal udføres (f.eks. sikkerhedstest, tilgængelighedstest)
- At bestemme omfanget af den test, der skal udføres
- At prioritere testen for at finde kritiske defekter så tidligt som muligt
- At bestemme, om der skal anvendes andre aktiviteter end test for at reducere risiko (f.eks. træning af uerfarne designere)

Risikobaseret test er afhængig af interessenternes fælles viden og indsigt for at kunne udføre en produktrisikoanalyse. Risikostyringsaktiviteterne giver en disciplineret tilgang til:

- Analyse og regelmæssig revurdering af risici
- Afgøre, hvilke risici, det er vigtigt at håndtere
- Implementere handlinger for at reducere disse risici
- Forberede en nødplan til at håndtere risici, hvis de skulle blive til virkelighed

Derudover kan test identificere nye risici, hjælpe med at afgøre, hvilke risici der skal reduceres og mindske usikkerheden for risici.

5.6 Defekthåndtering

Da en af målsætningerne ved test er at finde defekter, bør de defekter, der findes i test, logges. Den måde, defekter logges på kan variere afhængigt af den testede komponents eller systems kontekst, testniveauet og softwareudviklingens livscyklusmodel. Alle identificerede defekter bør undersøges og spores fra fund og klassifikation til løsningen (f.eks. udbedring af defekter og vellykket gentest af løsningen, udskydelse af efterfølgende release, accept af permanente begrænsninger i produktet osv.) For at kunne håndtere defekter frem mod løsningen, bør organisationen etablere defekthåndtering med et defineret workflow og regler for klassifi-

kation. Der skal være enighed om denne proces blandt alle, der deltager i defekthåndteringen, herunder arkitektere, designere, udviklere, testere og product owners. I nogle organisationer kan defektlogging og -sporing foregå meget uformelt.

I løbet af defekthåndteringsprocessen kan nogle af defektrapporterne vise sig at være falske positiver, ikke faktiske afvigelser pga. defekter. F.eks. kan en test fejle, hvis en netværksforbindelse er afbrudt eller har time-out. Falske positiver opstår ikke pga. en defekt i testobjektet, men er en anomali, der skal undersøges. Testerne bør forsøge at minimere antallet af falske positiver, der rapporteres som defekter.

Defekter kan rapporteres i løbet af kodning, statisk analyse, reviews, eller ved dynamiske test eller ved anvendelse af softwareproduktet. Defekter kan rapporteres for problemer i kode eller arbejdssystemer eller anden type dokumentation, herunder krav, userstories og acceptkriterier, udviklingsdokumenter, testdokumenter, brugervejledninger eller installationsvejledninger. For at have en effektiv og virkningsfuld defekthåndteringsproces, skal organisationen definere standarder for defekternes attributter, klassifikation og workflow.

En typisk defektrapport har følgende målsætninger:

- At give udviklere og andre parter oplysninger om negative hændelser, der fandt sted. Dette gøres for at lade dem identificere specifikke effekter, isolere problemet med en minimalt reproducerende test og for at rette potentielle defekter, hvis der er behov for det eller for at løse problemet
- At give testmanagere mulighed for at følge arbejdsproduktets kvalitet og kvalitetens effekt på testen. Eksempel: Hvis testerne rapporterer mange defekter, så har de brugt lang tid på at rapportere dem, og der vil derfor være behov for tid til at udføre yderligere test)
- At give idéer til forbedring af udviklings- og testprocessen

En defektrapport, der er indberettet under dynamisk test, omfatter typisk:

- En identifikation
- En titel og kort opsummering af den indrapporterede defekt
- Dato på defektrapporten, den udstedende organisation og forfatteren
- Identifikation af testelementet (konfigurationselement, der testes) og -miljøet
- De af udviklingslivscyklussens faser, hvori defekten blev observeret
- En beskrivelse af defekten, så den kan reproduceres og rettes, herunder logs, skærmbilleder af database og optagelser af testen
- Forventede og faktiske resultater
- Defektens omfang eller effektgrad (alvorlighed) på interessenters interesseområde
- Alvorsgrad/prioritering for udbedring
- Defektrapportens tilstand (f.eks. åben, udskudt, dublet, venter på udbedring, venter på gentest, genåbnet, lukket)
- Konklusioner, anbefalinger og akkrediteringer
- Globale problemer, hvor andre områder kan være påvirket af ændringer fra defekten
- Ændringshistorie, f.eks. hvad projektteamets medlemmer har gjort for at isolere og reparere defekten og bekræfte, at den er udbedret
- Referencer, herunder den testcase, der afslørede problemet

Nogle af disse detaljer, kan blive inkluderet og/eller styret automatisk, når man anvender defekthåndteringsværktøjer f.eks. automatisk tildeling af identifikation, tildeling og opdatering af defektrapportstatus i løbet af workflowet f.eks. Defekter fundet ved statisk test og specifikke reviews bliver normalt dokumenteret på en anden måde f.eks. i referater fra reviews.

Et eksempel på en defektrapports indhold kan findes i ISO-standard (ISO/IEC/IEEE 29119-3) (hvilket henviser til defektrapporter som hændelsesrapporter).

6. Værktøjsstøtte til test — 40 minutter

Nøgleord

datadrevet test, nøgleordsdrevet test, testafviklingsværktøj, testautomatisering, teststyringsværktøj

Læringsmål for testværktøjer

6.1 Overvejelser om testværktøj

FL-6.1.1 (K2) Klassificer testværktøjer efter deres formål og de testaktiviteter, de understøtter
FL-6.1.2 (K1) Identificer fordele og risici ved testautomatisering
FL-6.1.3 (K1) Husk særlige overvejelser i forbindelse med testafvikling og teststyringsværktøjer

6.2 Effektiv anvendelse af værktøjer

FL-6.2.1 (K1) Identificer hovedprincipper for valg af værktøj
FL-6.2.2 (K1) Husk målsætninger for anvendelse af pilotprojekter til at introducere værktøjer
FL-6.2.3 (K1) Identificer succeskriterier for evaluering, implementering, ibrugtagning og fortsat support af testværktøjer i en organisation

6.1 Overvejelser om testværktøj

Testværktøj kan anvendes til at understøtte en eller flere testaktiviteter. Disse værktøjer omfatter:

- Værktøjer, der anvendes direkte til test, som f.eks. testafviklingsværktøjer og testdataforberedelsesværktøjer
- Værktøjer, der hjælper med at håndtere krav, testcases, testproducerer, automatiserede testscripts, testdata og defekter, og til at rapportere og overvåge testafvikling
- Værktøjer, der anvendes til at analysere og evaluere
- Alle værktøjer, der hjælper i test (et regneark regnes også som et testværktøj i denne forstand)

6.1.1 Klassifikation af testværktøj

Testværktøj kan have et eller flere af de følgende formål, afhængigt af konteksten:

- At forbedre testaktiviteternes effektivitet ved at automatisere gentagne opgaver eller de opgaver, der kræver betydelige resurser, når de udføres manuelt (f.eks. testafvikling, regressionstest)
- At forbedre testaktiviteternes effektivitet ved at understøtte manuelle testaktiviteter igennem test-processen (se afsnit 1.4)
- At forbedre testaktiviteternes kvalitet ved at sikre mere konsistente tests og et højere niveau af reproducerbarhed for defekter
- At automatisere aktiviteter, der ikke kan afvikles manuelt (f.eks. performancetest i stor skala)
- At forøge testens pålidelighed (f.eks. ved at automatisere større datasammenligninger og simulation af adfærd)

Værktøj kan klassificeres på baggrund af forskellige kriterier som formål, pris, licensmodel (f.eks. kommerciel eller open source) og anvendt teknologi. Værktøj klassificeres i denne syllabus ud fra de testaktiviteter, de understøtter.

Nogle værktøjer understøtter kun, eller primært, én aktivitet; andre kan understøtte mere end en aktivitet, men vil blive klassificeret under den aktivitet, de primært er tilknyttet. Værktøjer fra en enkelt leverandør kan leveres som en integreret suite.

Nogle typer testværktøj kan være invaderende, hvilket betyder, at de påvirker testens resultat. F.eks. kan en applikationer faktiske svartid afvige pga. yderligere instruktioner, der udføres af et performancetestværktøj, eller kodedækning kan blive forvrænget af dækningsmåleværktøjet. Konsekvensen ved at anvende invaderende værktøjer kaldes undersøgelseseffekten.

Nogle værktøjer tilbyder support, der typisk er mest anvendelig for udviklere (f.eks. værktøjer, der anvendes i løbet af komponent- og integrationstest). Sådanne værktøjer står markeret med et "(U)" i afsnittene nedenfor.

Værktøjsstøtte til styring af testen og testware
Styringsværktøjer kan anvendes på alle testaktiviteter i hele softwareudviklingens livscyklus. Eksempler på værktøjer, der understøtter håndtering af at teste og testware:

- Teststyringsværktøjer og Application Lifecycle Management-værktøjer (ALM)
- Kravstyringsværktøj (f.eks. sporbarhed for testelementer)
- Defektstyringsværktøjer
- Konfigurationsstyringsværktøjer
- Continuous Integration-værktøjer (U)

Værktøjsstøtte til statisk test
Værktøjer til statisk test er beskrevet i afsnit 3. Eksempler på et sådanne værktøj:

- Statiske analyseværktøjer (U)

Værktøjsstøtte til testdesign og -implementering
Testdesignværktøjer hjælper til under udviklingen af vedligeholdelsesvenlige arbejdsprodukter i testdesign og -implementering, herunder testcases, testprocessen og testdata. Eksempler på sådanne værktøjer:

- Modelbaserede testværktøjer
- Testdataforberedelsesværktøjer

I nogle tilfælde kan værktøjer, der understøtter testdesign og -implementering også understøtte testafvikling og -logning eller kan levere output direkte til andre værktøjer, der understøtter testafvikling og -logning.

Værktøjsstøtte til testafvikling og -logning
Der findes mange værktøjer til at understøtte og forbedre testafviklings og -logningsaktiviteter. Eksempler på sådanne værktøjer:

- Testafviklingsværktøjer (f.eks. til at afvikle regressionstests)
- Dækningsmåleværktøjer (f.eks. kravdækning, kodedækning) (U)
- Testharness (U)

Værktøjsstøtte til performancemålinger og dynamiske analyser
Performancemålinger og dynamiske analyseværktøjer er vigtige til at understøtte præstation og belastnings-testaktiviteter, da disse aktiviteter ikke kan udføres effektivt manuelt. Eksempler på sådanne værktøjer:

- Performancetestværktøjer
- Dynamiske analyseværktøjer (U)

Værktøjsstøtte for specialiserede testbehov
Ud over værktøjer, der understøtter den generelle testproces, findes der mange andre værktøjer, der understøtter mere specifikke test for ikke funktionelle karakteristikker.

6.1.2 Fordele og risici ved testautomatisering

Det er ingen garanti for succes at en organisation anskaffer sig værktøj til testautomatisering. Hvert nyt værktøj, der introduceres i en organisation, kræver en indsats for at opnå reelle og vedvarende fordele. Der er potentielle fordele og muligheder forbundet med anvendelse af værktøjer i test, men der er også risici. Det er især tilfældet for testafviklingsværktøjer (der ofte kaldes testautomatisering).

Potentielle fordele ved at anvende værktøjer til at understøtte testafvikling:

- Reduktion i gentaget, manuelt arbejde (f.eks. afvikling af regressionstests, opgaver til opsætning/nedbrydning af testmiljø, genindtastning af samme testdata og sammenligning af kodestandarder) og dermed besparelser på tidsforbruget
- Høj konsistens og gentagelighed (f.eks. at testdata laves på en sammenhængende måde, tests bliver afviklet af et værktøj i samme rækkefølge med samme frekvens, og tests udledes konsekvent fra kravene)
- Mere objektive vurderinger (f.eks. statiske målinger, dækning)
- Nem adgang til oplysninger om test (f.eks. statistikker og grafer om testfremdrift, defektrater og performance)

Potentielle risici ved at anvende værktøjer til at understøtte test:

- Forventninger til værktøjet kan være urealistisk høje (herunder funktionalitet og hvor let det er at anvende)
- Man kan undervurdere tid, omkostninger og indsats i forbindelse med introduktion af et værktøj (herunder træning og ekstern ekspertise)
- Man kan undervurdere den nødvendige tid og indsats for at opnå betydelige og vedvarende fordele fra værktøjet (herunder behov for ændringer i testprocessen og vedvarende forbedringer i den måde, værktøjet anvendes på)
- Man kan undervurdere den påkrævede indsats for at opretholde de testarbejdsprodukter, der genereres af værktøjet
- Man kan gøre sig for afhængig af værktøjet (værktøjet opfattes som en erstatning for testdesign eller -afvikling eller bliver anvendt til automatiseret test, hvor manuel test ville have været bedre)
- Versionskontrol af testarbejdsprodukter kan blive forsømt
- Der kan opstå problemer i samspillet mellem kritiske værktøjer, f.eks. kravstyringsværktøjer, konfigurationsstyringsværktøjer, defektstyringsværktøjer og værktøjer fra forskellige leverandører
- Værktøjsleverandøren kan gå konkurs, standse support af værktøjet eller sælge værktøjet til en anden leverandør
- Leverandøren kan yde ringe support, mangelfulde opgraderinger eller mangelfuld udbedring af defekter
- Et open source-projekt kan stoppe
- Værktøjet viser sig ikke at understøtte en ny platform eller teknologi
- Værktøjets anvendelsesområde kan være uklart defineret (f.eks. til mentorordning, opdatering osv.)

6.1.3 Særlige overvejelse ved testafvikling og teststyringsværktøjer

For en nem og vellykket implementering er der en række forhold, der skal overvejes, når man vælger testafviklings- og teststyringsværktøjer og integrerer dem i en organisation.

Testafviklingsværktøjer
Testafviklingsværktøjer afvikler testelementer vha. automatiserede testscripts. Denne type værktøj kræver ofte en væsentlig indsats for at opnå fordele af betydning.

- **Opdagende testtilgang:** Det kan være tillokkende at optage en testers manuelle handlinger, men denne tilgang giver ingen fordele ved et stort antal testscripts. Et optaget script er en lineær repræsentation med specifikke data og handlinger, som en del af hvert script. Denne type scripts kan vise sig at være ustabile, når brugerfladen bliver ændret og kræver vedvarende vedligeholdelse, da systemets brugerflade løbende udvikles
- **Datadrevet testtilgang:** Denne testtilgang anvender et generisk testscript, der kan læse inputdata og afvikle samme testscript med forskellige data
- **Nøgleordsdrevet testtilgang:** Denne testtilgang behandler et generisk script nøgleord, der beskriver de handlinger, som skal foretages (der også kaldes handlingsord), og henter så nøgleordsscripts til at behandle tilknyttet testdata

De ovenstående tilgange kræver nogen med ekspertise i scriptsprog (testere, udviklere eller specialister i testautomatisering). Ved at bruge en datadrevet eller nøgleordsdrevet testtilgang kan testere som ikke kender scriptsproget også bigrade ved at skabe testdata og/eller nøgleord til de prædefinerede scripts. Uanset hvilken scriptteknik, der anvendes, skal det forventede resultat sammenlignes med det faktiske resultat fra testen, enten dynamisk (mens testen afvikles) eller senere (post-afvikling) sammenligning.

Yderligere detaljer og eksempler på datadrevne og nøgleordsdrevne testtilgange står beskrevet i ISTQB® CTAL-TAE Certified Tester Advanced Level Test Automation Engineer syllabus, Fewster 1999 og Buwalda 2001.

Værktøjer til modelbaseret test (MBT) muliggør en funktionel specifikation, der kan optages i form af en model, som f.eks. et aktivitetsdiagram. Denne opgave udføres i almindelighed af en systemdesigner. MBT-værktøjet fortolker modellen for at danne testcase-specifikationer, der kan gemmes i et teststyringsværktøj og/eller afvikles af et testafviklingsværktøj (se ISTQB® CTFL-MBT Certified Tester Foundation Level Model-Based Testing syllabus).

Teststyringsværktøjer
Det er ofte nødvendigt for teststyringsværktøjer at arbejde sammen med andre værktøjer eller regneark af forskellige årsager, herunder:

- For at producere anvendelige oplysninger i et format, der passer til organisationens behov
- For at opretholde konsekvent sporbarhed, der følger kravene i et kravstyringsværktøj
- For at bidrage med testobjektoplysninger i konfigurationsstyringsværktøj

Det er særligt vigtigt at overveje, når et integreret værktøj (f.eks. applikationslivscyklusstyring) anvendes med andre grupper i en organisation. Især hvis det omfatter et teststyringsmodul, såvel som andre moduler (f.eks. projektets tidsplan og budgetoplysninger).

6.2 Effektiv anvendelse af værktøjer

6.2.1 Hovedprincipper for valg af værktøj

Hovedovervejelserne ved at vælge et værktøj til en organisation:

- At vurdere ens egen organisations modenhed, styrker og svagheder
- At identificere mulighederne for en forbedret testproces understøttet af værktøjer
- At forstå de teknologier, der anvendes af testelementet for at vælge et værktøj, der er kompatibelt med teknologien
- At forstå værktøjets kompatibilitet og integration i forhold til det build og Continuous Integration værktøjer, der allerede anvendes inden for organisationen
- At evaluere værktøjet i forhold til tydelige krav og objektive kriterier
- At overveje, om værktøjet er tilgængeligt i en gratis prøveperiode (og i hvor lang tid)
- At evaluere leverandøren (herunder træning, support og kommercielle aspekter) og understøttelse af ikke-kommercielle (f.eks. open source) værktøjer
- At identificere interne krav til coaching og mentoring ved anvendelsen af værktøjer
- At evaluere træningsbehov for dem, der skal arbejde direkte med værktøjets færdigheder inden for test (og testautomatisering)
- At overveje fordele og ulemper ved forskellige licensmodeller (f.eks. kommercielle eller open source)
- At vurdere cost-benefit forhold baseret på den konkrete business case

Endelig bør man udføre en proof-of-concept evaluering for at afgøre, om værktøjet fungerer som det skal med softwaren under test i den nuværende infrastruktur. Hvis det er nødvendigt, må man identificere de nødvendige ændringer for at værktøjet kan anvendes effektivt.

6.2.2 Pilotprojekter til at introducere et værktøj i en organisation

Efter at organisationen har valgt et værktøj, skal det introduceres som et pilotprojekt med følgende målsætninger:

- At opnå dybdegående viden om værktøjet og at forstå dets styrker og svagheder
- At evaluere værktøjet i forhold til eksisterende procedurer og praksisser, og afgøre, hvad der skal ændres
- At afgøre standardmetoder for anvendelse, håndtering, lagring og vedligeholdelse af værktøj og test-arbejdsprodukter (f.eks. vælge navngivningskonventioner for filer og tests, vælge kodestandarder, opbygge biblioteker og definere de forskellige testsuites' modularitet)
- At vurdere om fordelene kan opnås inden for en rimelig pris
- At forstå den metrik, som værktøjet skal indsamle og rapportere, og at konfigurere værktøjet for at sikre, at disse metrikker kan optages og rapporteres

6.2.3 Succesfaktorer for værktøjer

Succesfaktorer for evaluering, implementering, ibrugtagning og vedvarende support af værktøjer inden for en organisation:

- Trinvis udrulning af værktøjet til resten af organisationen
- Tilpasse og forbedre processen, så den passer til anvendelsen af værktøjet
- Tilbyde træning, coaching og mentoring til værktøjets brugere
- Definere retningslinjer for anvendelsen af værktøjet (f.eks. interne standarder for automatisering)
- Implementere en måde at indsamle brugbare oplysninger på fra den faktiske anvendelse af værktø-jet
- Overvåge anvendelsen af værktøjet og dets fordele
- Yde support til værktøjsbrugerne
- Indsamle viden fra brugerne

Det er vigtigt at sikre, at værktøjet er teknisk og organisatorisk integreret i softwareudviklingslivscyklussen, hvilket kan betyde sammenhænge, hvor forskellige organisationer er ansvarlige for drift og/eller tredjeparts-leverance.

Se Graham 2012 for erfaringer og råd vedrørende anvendelse af testafviklingsværktøjer.

7. Henvisninger

Standarder

- ISO/IEC/IEEE 29119-1 (2013) Programmel- og systemudvikling - Softwaretest – del 1: Koncepter og definitioner
- ISO/IEC/IEEE 29119-2 (2013) Programmel- og systemudvikling - Softwaretest – del 2: Testprocedurer
- ISO/IEC/IEEE 29119-3 (2013) Programmel- og systemudvikling - Softwaretest – del 3: Testdokumentation
- ISO/IEC/IEEE 29119-4 (2015) Programmel- og systemudvikling - Softwaretest – del 4: Testteknikker
- ISO/IEC 25010, (2011) System- og programmeludvikling – System- og softwarekvalitetskrav og evaluering (SQuaRE) System og softwarekvalitetsmodeller
- ISO/IEC 20246: (2017) Programmel- og systemudvikling – Reviews af arbejdsprodukter
- UML 2.5, Unified Modeling Language referencemanual, http://www.omg.org/spec/UML/2.5.1/, 2017

ISTQB® dokumenter

- ISTQB® begrebsliste
- ISTQB® CTFL Certified Tester Foundation Level 2018
- ISTQB® CTFL-MBT Certified Tester Foundation Level Model-Based Tester syllabus
- ISTQB® CTFL-AT Certified Tester Foundation Level Agile Tester syllabus
- ISTQB® CTAL-TA Certified Tester Advanced Level Test Analyst syllabus
- ISTQB® CTAL-TM Certified Tester Advanced Level Test Manager syllabus
- ISTQB® CTAL-SEC Certified Tester Advanced Level Security Tester syllabus
- ISTQB® CTAL-TAE Certified Tester Advanced Level Test Automation Engineer syllabus
- ISTQB® CTEL-TM Certified Tester Expert Level Test Management syllabus
- ISTQB® CTEL-ITP Certified Tester Expert Level Improving the Test Process syllabus

Bøger og artikler

- Beizer, B. (1990) Software Testing Techniques (2e), Van Nostrand Reinhold: Boston MA
- Black, R. (2017) Agile Testing Foundations, BCS Learning & Development Ltd: Swindon UK
- Black, R. (2009) Managing the Testing Process (3e), John Wiley & Sons: New York NY
- Buwalda, H. et al. (2001) Integrated Test Design and Automation, Addison Wesley: Reading MA
- Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: Norwood MA
- Craig, R. and Jaskiel, S. (2002) Systematic Software Testing, Artech House: Norwood MA
- Crispin, L. and Gregory, J. (2008) Agile Testing, Pearson Education: Boston MA
- Fewster, M. and Graham, D. (1999) Software Test Automation, Addison Wesley: Harlow UK
- Gilb, T. and Graham, D. (1993) Software Inspection, Addison Wesley: Reading MA
- Graham, D. and Fewster, M. (2012) Experiences of Test Automation, Pearson Education: Boston MA
- Gregory, J. and Crispin, L. (2015) More Agile Testing, Pearson Education: Boston MA
- Jorgensen, P. (2014) Software Testing, A Craftsman's Approach (4e), CRC Press: Boca Raton FL
- Kaner, C., Bach, J. and Pettichord, B. (2002) Lessons Learned in Software Testing, John Wiley & Sons:
- New York NY
- Kaner, C., Padmanabhan, S. and Hoffman, D. (2013) The Domain Testing Workbook, Context-Driven Press: New York NY
- Kramer, A., Legeard, B. (2016) Model-Based Testing Essentials: Guide to the ISTQB® Certified Model-Based Tester: Foundation Level, John Wiley & Sons: New York NY
- Myers, G. (2011) The Art of Software Testing, (3e), John Wiley & Sons: New York NY
- Sauer, C. (2000) "The Effectiveness of Software Development Technical Reviews: A Behaviorally
- Motivated Program of Research," IEEE Transactions on Software Engineering, Volume 26, Issue 1, pp 1-

- Shull, F., Rus, I., Basili, V. July 2000. “How Perspective-Based Reading can Improve Requirement Inspections.” IEEE Computer, Volume 33, Issue 7, pp 73-79
- van Veenendaal, E. (ed.) (2004) The Testing Practitioner (Chapters 8 - 10), UTN Publishers: The Netherlands
- Wiegers, K. (2002) Peer Reviews in Software, Pearson Education: Boston MA
- Weinberg, G. (2008) Perfect Software and Other Illusions about Testing, Dorset House: New York NY

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

- Black, R., van Veenendaal, E. and Graham, D. (2019) Foundations of Software Testing: ISTQB® Certification (4e), Cengage Learning: London UK
- Hetzel, W. (1993) Complete Guide to Software Testing (2e), QED Information Sciences: Wellesley MA

8. Bilag A – Baggrund for syllabus

Dette dokuments historie

Dette dokument er den danske oversatte syllabus af ISTQB® Certified Tester Foundation Level syllabus, godkendt af ISTQB® (www.istqb.org).

Dette dokumenter (på engelsk) blev udarbejdet mellem juni og august 2019 af en arbejdsgruppe bestående af medlemmer udvalgt af International Software Testing Qualifications Board (ISTQB®). Opdateringen blev lavet på baggrund af indsendte reviewkommentarer fra lokale ISTQB® medlemmer på baggrund af deres erfaring og brug af Certified Tester syllabus 2018.

ISTQB® Certified Tester Foundation Level syllabus (engelsk udgave) blev udarbejdet mellem 2014 og 2018 af en arbejdsgruppe bestående af medlemmer, udpeget af International Software Testing Qualifications Board (ISTQB®). 2018-versionen blev først reviewet af et reviewpanel udvalgt blandt lokale ISTQB® medlemmer, og siden af repræsentanter fra det internationale softwaretestsamfund.

Målsætninger for Foundation Certificate-kvalifikation

- At opnå anerkendelse af test som en væsentlig og professionel softwareteknisk specialisering
- At levere en standardmodel til udvikling af testeres karriere
- At sørge for, at professionelt kvalificerede testere bliver anerkendt af arbejdsgivere, kunder og kolleger, og til at hæve testeres profil
- At fremme konsistent og god testpraksis inden for alle softwaretekniske discipliner
- At identificere testemner, der er relevante og værdifulde for industrien
- At sætte softwareleverandører i stand til at ansætte godkendte testere og dermed få en kommerciel fordel over deres konkurrenter ved at reklamere for deres ansættelsespolitik
- At skabe en mulighed for testere og de, der har interesse i test, til at opnå en internationalt anerkendt kvalifikation i emnet

Målsætninger for international kvalifikation

- At blive i stand til at sammenligne testevner på tværs af forskellige lande
- At sætte testere i stand til lettere at bevæge sig på tværs af landegrænserne
- At gøre det muligt for multinationale eller internationale projekter at opnå fælles forståelse for testemnerne
- At øge antallet af kvalificerede testere verden over
- At opnå større effekt/værdi som et internationalt baseret initiativ i stedet for en landespecifik metode
- At udvikle en fælles international enhed med forståelse og kendskab til test via denne syllabus og terminologi, og til at øge kendskabsniveauet for test for alle deltagere
- At fremme test som et erhverv i flere lande
- At sætte testere i stand til at opnå en anerkendt kvalifikation på deres eget sprog
- At gøre det muligt at dele kendskab og ressourcer på tværs af lande
- At opnå international anerkendelse af testere og denne kvalifikation igennem deltagelse fra mange lande

Adgangskrav for denne kvalificering

Adgangskriteriet for at tage ISTQB® Certified Tester Foundation Level eksamen er, at kandidaten har en interesse i softwaretest. Det anbefales imidlertid på det kraftigste, at kandidaten også:

- Som minimum har lidt baggrund i enten softwareudvikling eller softwaretest f.eks. seks måneders erfaring som system- eller brugeraccepttester eller som softwareudvikler
- Tager et kursus, der er godkendt til ISTQB®-standarder (af et af de ISTQB®-anerkendte nationale nævn)

Baggrund og historie for Foundation Certificate i softwaretest

Den uafhængige certificering af softwaretestere begyndte i Storbritannien hos British Computer Society's Information Systems Examination Board (JSEB), da der blev etableret et softwaretestnævn i 1998 (www.bcs.org.uk/iseb). I 2002 begyndte ASQF i Tyskland at arbejde for en tysk kvalifikationsordning for testere (www.asqf.de). Denne syllabus er baseret på ISEB- og ASQF-syllabierne; den omfatter både reorganiseret, opdateret og noget nyt indhold, og der er lagt målrettet vægt på emner, der vil give den mest praktiske hjælp til testere.

Et eksisterende Foundation Certificate (f.eks. fra ISEB, ASQF eller et ISTQB®-anerkendt nationalt nævn), der er godkendt, inden International Certificate frigives, anses for at svare til International Certificate. Foundation bliver ikke forældet og behøver ikke at blive fornyet. Datoen, da det blev udstedt, står skrevet på certifikatet.

I hvert af de deltagende lande bliver lokale forhold kontrolleret af et nationalt ISTQB®-anerkendt softwaretestnævn. De nationale nævns opgaver er specificeret af ISTQB®, men er implementeret i hvert enkelt land. Opgaverne for de lokale ISTQB® medlemmer forventes at omfatte akkreditering af kursusudbydere og afholdelse af eksamen.

9. Bilag B – Læringsmål/Kognitivt vidensniveau

Følgende undervisningsmål er defineret som gældende for denne syllabus. Der bliver eksamineret i alle emner i syllabus i henhold til deres undervisningsmål.

Niveau 1: Huske (K1)

Kandidaten skal kunne kende, huske og nævne en term eller et koncept.

Nøgleord: Identificere, huske, hente, nævne, genkende og vide

Eksempler:
Kan huske definitionen af en "afvigelse" som:

- "Manglende levering af service til en slutbruger eller enhver anden interessent" eller
- "Komponenten eller systemet viser afvigende opførsel fra forventet levering, service eller resultat"

Niveau 2: Forstå (K2)

Kandidaten kan udvælge årsager eller forklaringer for instruktioner, relateret til emnet, og kan opsummere, sammenligne, klassificere, kategorisere og give eksempler på testkonceptet.

Nøgleord: Abstract, eksemplificere, generalisere, kategorisere, klassificere, konkludere, konstruere modeller, kortlægge, opsummere, repræsentere, sammenligne, tolke, udlede

Eksempler:
Kan forklare årsagen til, at analyse og design af tests bør finde sted så tidligt som muligt:

- At finde defekter, når de er billigere at fjerne
- At finde de vigtigste defekter først

Kan forklare ligheder og forskelle mellem integrations- og systemtest:

- Ligheder: testobjekterne både i integrationstest og systemtest omfatter mere end en komponent, og både integrationstest og systemtest kan omfatte ikke-funktionelle aspekter
- Forskelle: integrationstest fokuserer på brugergrænseflader og interaktioner, og systemtest fokuserer på forhold for hele systemet, som f.eks. start-til-slut bearbejdning

Niveau 3: Anvende (K1)

Kandidaten kan udvælge den korrekte anvendelse eller teknik og anvende den i en given kontekst.

Nøgleord: Afvikle, anvendelse af en procedure, anvende, følge en procedure, implementere

Eksempler:

- Kan identificere grænseværdier for gyldige og ugyldige partitioner
- Kan vælge testcases ud fra et givent tilstandsovergangsdiagram for at dække alle overgange

Henvisninger (For læringsmålenes kognitive niveauer)
Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon: Boston MA

10. Bilag C – Udgivelsesnoter

ISTQB® Certified Tester syllabus 2018 v3.1 er en mindre opdatering af syllabus 2018 (v3.0). Der er udgivet en release note med et overblik over indholdet at hvert kapitel samt en version med track changes i forhold til syllabus 2018.

ISTQB® CTFL Certified Tester Foundation Level syllabus 2018 er en større opdatering og omskrivning af 2011-frigivelsen. Derfor er der ingen detaljerede release noter for hvert kapitel og afsnit. En opsummering af de vigtigste ændringer kan dog findes her. Derudover beskriver ISTQB® sporbarheden i det separate dokument med frigivelsesnoter mellem læringsmål i 2011-versionen af Certified Tester Foundation Level syllabus og læringsmålene i 2018-versionen, hvor det kan ses, hvilke tilføjelser, opdateringer og udeladelse, der er foretaget.

I begyndelsen af 2017 havde mere end 550.000 personer i mere end 100 lande taget Certified Tester Foundation Level eksamen, og mere end 500.000 personer verden over er certificeret. Med en forventning om, at man skal have læst Certified Tester Foundation Level syllabus for at bestå eksamen, gør det sandsynligt, at Certified Tester Foundation Level syllabus er det mest læste softwaretestdokument nogensinde!

Denne større opdatering udføres i henhold til denne arv og for at forbedre den værdi, som ISTQB® leverer til de næste 500.000 personer i det globale testsamfund.

I denne version er alle læringsmålene blevet ændret for at gøre dem atomare, og for at oprette en tydelig sporbarhed mellem alle af læringsmålene i hvert afsnit (og eksamensspørgsmålene), der er relateret til det pågældende læringsmål, og at have en tydelig sporbarhed fra afsnittene (og eksamensspørgsmålene) tilbage til de tilknyttede læringsmål. Derudover er tildelingen af tid for hvert kapitel gjort mere realistisk i forhold til 2011-versionen af syllabus ved at anvende gennemprøvet heuristik og formler sammen med andre ISTQB®-certificeringer, der er baseret på en analyse af de læringsmål, som er dækket i hvert kapitel.

Selvom dette er et Foundation Level syllabus, der beskriver de bedste praksisser og teknikker, som har formået at bestå igennem tiden, har vi foretaget ændringer for at modernisere præsentationen af materialet, særligt i forhold til softwareudviklingsmetoder (f.eks. Scrum og Continuous Deployment) og teknologier (f.eks. Internet of Things). Vi har opdateret de henviste standarder for at gøre dem mere tidssvarende på følgende måde:

1. ISO/IEC/IEEE 29119 erstatter IEEE Standard 829
2. ISO/IEC 25010 erstatter ISO 9126
3. ISO/IEC 20246 erstatter IEEE1028

Derudover, da ISTQB®-porteføljen er vokset dramatisk over det seneste årti, har vi tilføjet omfattende krydsreferencer til relateret materiale i andre ISTQB®-syllabi, der hvor det er relevant, såvel som at reviewe overensstemmelser mellem alle syllabier og ISTQB®s ordliste. Målet er at gøre denne udgave nemmere at læse, forstå, lære og oversætte med fokus på at øge den praktiske anvendelighed og balancen mellem viden og færdigheder.

For en detaljeret analyse af de ændringer, som er foretaget til denne frigivelse, se ISTQB® Certified Tester Foundation Level Overview 2018.

11. Indeks