Sertifisert Tester

Pensum for grunnivå

(Foundation Level)

Norsk versjon basert på CTFL 2018

V1.0

28.04.2019

Norwegian Testing Board

International Software Testing Qualifications Board

Revisjonshistorie

Versjon	Dato	Bemerkninger
Alfa versjon	21.01.2019	Etter oversettelse
Beta versjon	30.03.2019	Etter alfa review
1.0	28.04.2019	Første versjon

Innholdsfortegnelse

Engelsk originalversjon

Forfatterne av engelsk versjon 2018: Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (project manager), Debra Friedenberg, Matthias Hamburg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, og Eshraka Zakaria

Oversettelsen til norsk

Oversettelsen til norsk er utført av medlemmer i Norwegian Testing Board (som er det nasjonale ISTQB board i Norge). Disse er spesialister i testing. Oversettelsen er gjort på dugnad.

Oversettelsen holder seg tett til den engelske originalen. Fordi denne blir brukt i den internasjonale sertifiseringen, vil vi sikre at oversettelsen dekker alle aspekter og nyanser ved originalen. Vi er klar over at språket derfor delvis virker noe tungt.

Vi er takknemlige for all tilbakemelding. Vi prøver også å besvare spørsmål du måtte ha innen rimelig tid. Kommentarer kan sendes til leder@istqb-norge.no.

0 Innledning

0.1 Formål med dette dokument

Dette pensum er grunnlaget for International Software Testing Qualification på grunnivå (Foundation Level).

ISTQB publiserer dette pensum med følgende formål:

1. Til de nasjonale Boards, for oversettelse til sitt lokale sprak og akkreditering av kursleverandører. Nasjonale Boards kan tilpasse pensum til spesielle språkbehov og legge til referanser til lokale publikasjoner
2. Til sertifiseringsorganer, for utarbeidelse av eksamensspørsmål på sitt lokale språk tilpasset læringsmålene for denne pensum
3. Til kursholdere, for å produsere kursmateriale og vurdere hensiktsmessige læringsmetoder
4. Til kandidater, for å forberede seg til sertifiseringseksamen (enten som en del av et kurs eller uavhengig)
5. Til det internasjonale programvare- og systemteknologifellesskap, for å fremme testyrket og som grunnlag for bøker og artikler

ISTQB kan tillate andre organisasjoner og bedrifter å bruke dette pensum til andre formål, forutsatt at de søker og får skriftlig tillatelse fra ISTQB på forhånd.

0.2 Sertifisert tester på grunnivå (Foundation Level) i testing

Målgruppen for sertifiseringen på grunnivå (Foundation Level) er enhver som arbeider med test av programvare. Dette omfatter personer i roller som testere, testutviklere, testingeniører, testkonsulenter, testledere, brukere som akseptansetestere og programvareutviklere. Sertifiseringen på grunnivå passer også for den som vil ha en grunnleggende forståelse av programvaretesting, som f.eks. produkteiere, prosjektledere, kvalitetsledere, utviklingsledere for programvare, forretningsanalytikere, IT-direktører og ledelseskonsulenter. De som innehar dette sertifikat kan gå videre til testsertifisering på høyere nivå.

ISTQB Foundation Level Overview 2018 (eng) er et separat dokument som presenterer følgende informasjon:

- Nytteverdien av pensum for bedriften
- Tabell som viser sporbarhet mellom nytteverdiene og læremål
- Oppsummering av dette pensum

0.3 Læremål og kunnskapsnivå

Læremål brukes til å produsere «Certified Tester Foundation Level» (CTFL) eksamen.

Generelt kan alt innholdet i dette pensum bli eksaminert på K1-nivå, bortsett fra introduksjonen og vedleggene. Det vil si at kandidaten kan bli bedt om å gjenkjenne, huske eller gjengi et nøkkelord, begrep eller konsept nevnt i noen av de seks kapitlene. Kunnskapsnivåene for de spesifikke læremål er vist i begynnelsen av hvert kapittel og klassifisert som følger:

- K1: huske
- K2: forstå
- K3: bruke

Videre detaljer og eksempler av læremål er gitt i vedlegg B.

Betydningen av alle uttrykk under overskriften “Nøkkelord” rett under hver kapitteloverskrift skal huskes (K1), selv om de ikke blir uttrykkelig nevnt i læremålene.

0.4 Sertifiseringseksamen

Eksamen vil være basert på dette pensum. Svar på eksamensspørsmål kan kreve bruk av materiale fra flere steder i dette pensum. Alle deler av pensum kan eksamineres med unntak av introduksjonen og vedleggene. Standarder, bøker og andre ISTQB-pensum er inkludert som referanser, men innholdet er ikke påkrevd, utover det som er oppsummert i dette pensum.

Eksamensformatet er flervalgseksamen. Det er 40 spørsmål. For å bestå eksamen må minst 65% av spørsmålene (dvs. 26 spørsmål) besvares riktig.

Eksamen kan tas som en del av et akkreditert kurs eller uten kurs, f.eks. på et eksamenssenter eller i en offentlig eksamen. Kurs er altså ikke en forutsetning for å kunne ta eksamen.

0.5 Akkreditering

Kursholdere og deres kursmateriale som følger dette pensum, kan akkrediteres av Norwegian Testing Board (NTB). Retningslinjer for akkreditering bør hentes fra NTB. Et akkreditert kurs som er anerkjent i samsvar med dette pensum og som holdes av akkreditert instruktør, har krav på å få arrangert en ISTQB-eksamen som del av kurset.

0.6 Detaljnivå

Detaljnivået i dette pensum muliggjør internasjonalt sammenlignbare kurs og eksamen. For å få dette til, består dette pensum av:

- Generelle læremål som beskriver formålet med grunnivået (Foundation Level)
- Liste med nøkkelord som kandidatene må kunne gjenta og ha forstått
- Læremål for hvert kunnskapsområde, som beskriver hva folk skal kunne og forstå

Beskrivelse av nøkkelkonsepter som skal læres, inklusive kildereferanser slik som alminnelig akseptert litteratur og standarder.

Innholdet i pensum er ikke en beskrivelse av hele kunnskapsområdet testing av programvare. Det beskriver bare detaljnivået som skal dekkes i et grunnleggende kurs. Det fokuserer på testkonsepter og teknikker som kan gjelde for alle programvareprosjekter, inkludert smidige prosjekter. Dette pensumet inneholder ikke noen spesifikke læringsmål knyttet til en bestemt modell eller metode for programvareutvikling, men belyser hvordan disse begrepene gjelder i trinnvise og iterative og i sekvensielle utviklingsmodeller.

0.7 Hvordan dette pensum er organisert

Det er seks hovedkapitler som kan eksamineres. Kapitteloverskriften viser den minimale kurstiden for kapitlet. Tid er ikke videre spesifisert for de andre nivåer i pensumet. Det krever minst 16,75 timer undervisning for et akkreditert kurs fordelt over de 6 kapitlene:

Kapitel 1: Grunnleggende fakta om testing- 175 minutter

Kapitel 2: Testing i programvarens livssyklus - 100 minutter

Kapitel 3: Statisk testing - 135 minutter

Kapitel 4: Testteknikker - 330 minutter

Kapitel 5: Testledelse - 225 minutter

Kapitel 6: Verktøystøtte for testing - 40 minutter

0.8 Forklaring av spesiell terminologi på norsk

I en del situasjoner er det vanskelig å finne treffende begreper på norsk, eller det er brukt alternative formuleringer. I dette avsnitt er noen språklige problemer forklart.

Test vs. Testing: de to ordene er brukt med samme betydning i pensum.

Feil: på engelsk finnes begrepene error, mistake, fault, defect, problem, failure. På norsk bruker vi samme ord for alle disse begrepene, nemlig feil. For å kunne skille mellom årsaken til en feil, selve feilen og de følgene en feil har når den blir utført, brukes de engelske uttrykk i den norske eksamen.

Review, gjennomgang og gransking: disse ordene er brukt i samme betydning.

Test work product: vanskelig å oversette, det finnes ikke et norsk ord som er godt etablert og har samme betydningsbredden som det engelske begrepet. Oversatt til 'arbeidsresultat' hhv 'arbeidsprodukt', av og til 'dokument' (oftest er det snakk om dokumenter).

Software development lifecycle: oversatt med 'programvarens livssyklus', av og til kun referert med 'livssyklus' der det er innlysende at det dreier seg om programvare.

Software development lifecycle models: oftest oversatt med 'utviklingsmodeller'.

1 Grunnleggende fakta om testing — 175 minutter

Nøkkelord

debugging, eng: defect, eng: error, eng: failure, feil, feilkilde, kjøreplan, kvalitet, kvalitetssikring, sporbarhet, testanalyse, testavslutning, testbetingelse, testdata, testdekning, testdesign, testgrunnlag, testimplementering, testing, testmateriell, testmål, testobjekt, testorakel, testovervåking, testplanlegging, testprosedyre, teststyring, testsuite, testtilfelle, testutføring, validering, verifisering

Læremål:

1.1 Hva er testing?

FL-1.1.1 (K1) Identifiser typiske mål for testing

FL-1.1.2 (K2) Forklar forskjellen mellom testing og debugging

1.2 Hvorfor er testing nødvendig?

FL-1.2.1 (K2) Gi eksempler for hvorfor testing er nødvendig

FL-1.2.2 (K2) Beskriv sammenhengen mellom testing og kvalitetssikring og gi eksempler på hvordan testing bidrar til høyere kvalitet

FL-1.2.3 (K2) Skill mellom de engelske begrepene error, defect og failure

FL-1.2.4 (K2) Skill mellom feilkilde og virkninger av feil

1.3 Syv testprinsipper

FL-1.3.1 (K2) Forklar de syv testprinsippene

1.4 Testprosessen

FL-1.4.1 (K2) Forklar hvordan konteksten påvirker testprosessen

FL-1.4.2 (K2) Beskriv testaktiviteter og respektive oppgaver i testprosessen

FL-1.4.3 (K2) Forklar forskjellen mellom arbeidsresultatene som støtter testprosessen

FL-1.4.4 (K2) Forklar verdien av å vedlikeholde sporbarhet mellom testgrunnlag og arbeidsresultatene fra testen

1.5 Testingens psykologi

FL-1.5.1 (K1) Identifiser de psykologiske faktorene som har innflytelse på testingens suksess

FL-1.5.2 (K2) Forklar forskjellen mellom tankesettet som kreves for test og tankesettet som kreves for utvikling

1.1 Hva er testing?

Programvaresystemer er en integrert del av vårt liv, fra forretningsapplikasjoner (f.eks. bank) til forbrukerprodukter (f.eks. biler). De fleste har erfart at programvare ikke alltid virker som forventet. Programvare som ikke fungerer riktig kan føre til mange problemer, f.eks. tap av penger, tid eller anseelse, eller til og med skade på mennesker eller død. Testing av programvare er en måte å kontrollere programvarens kvalitet og tjener til å redusere risikoen for at programvaren feiler i drift.

En vanlig misforståelse er at testing bare består av å kjøre tester, dvs. å utføre programvaren og å sjekke resultatene. Som det er beskrevet i avsnitt 1.4, er testing en prosess som omfatter mange forskjellige aktiviteter. Relevante aktiviteter er å planlegge testingen, å analysere, designe og implementere tester, å rapportere testframdriften og testresultater og å evaluere kvaliteten til et testobjekt.

Testing kan omfatte å utføre komponenten eller systemet som blir testet. Slik testing blir kalt dynamisk testing. Test som ikke utfører noe blir kalt statisk testing. Testing omfatter altså også å gjennomgå eller granske arbeidsresultater som krav, brukshistorier og kildekode.

En annen vanlig misforståelse er at testing utelukkende fokuserer på verifisering av krav, brukshistorier og andre spesifikasjoner. Det er riktig at testing skal sjekke om systemet oppfyller spesifiserte krav. I tillegg omfatter testing også validering, dvs. å sjekke om systemet oppfyller behovene til brukere og andre interessenter i vanlig bruksomgivelse.

Testaktiviteter blir forskjellig organisert og utført i ulike livssyklusmodeller (se avsnitt 2.1).

1.1.1 Typiske mål for testing

For et gitt prosjekt kan testmål omfatte:

- Evaluere arbeidsresultater som krav, brukshistorier, design og kode
- Verifisere om alle spesifiserte krav er oppfylt
- Validere om testobjektet er fullstendig og virker som brukerne og de andre interessentene forventer
- Skaffe tiltro til testobjektets kvalitetsnivå
- Forebygge feil
- Finne feil og deres årsaker
- Skaffe nok informasjon til interessentene slik at de kan foreta velfunderte beslutninger, spesielt vedrørende kvalitetsnivået til testobjektet
- Redusere risikoen for dårlig programvarekvalitet (dvs. risikoen for at hittil uoppdagede feil vil skje i drift)
- Oppfylle kontraktsmessige, juridiske eller lovpålagte krav og forskrifter eller standarder, og/eller verifisere testobjektets samsvar med slike krav eller standarder

Mål for testingen kan variere, avhengig av konteksten til den komponenten eller systemet som testes, testnivået og livssyklusmodellen for programvaren. Forskjellene kan f.eks. omfatte:

- Under komponenttesting kan et mål være å finne så mange feil og feilsituasjoner (failures) som mulig, slik at en kan rette de underliggende feilkildene (defects) tidlig. Et annet mål kan være å øke kodedekning av komponenttestene

- Under akseptansetesting kan et mål være å bekrefte at systemet virker som forventet og oppfyller krav. Et annet mål av denne testingen kan være å gi informasjon til interessenter om risikoen ved å ta i bruk systemet ved et gitt tidspunkt

1.1.2 Testing og debugging

Testing og debugging er forskjellig. Å utføre tester kan vise feilsituasjoner (failures) som er forårsaket av feil i programvaren (defects). Debugging er utviklingsaktiviteten som finner, analyserer og retter slike feil (defects). Etterfølgende retesting kontrollerer om rettelsene virkelig har fjernet feilene. I noen tilfeller er testerne ansvarlig for den første testen og den etterfølgende retesten, mens utviklere gjør debuggingen og den tilhørende komponenttestingen. Testere kan også delta i debugging og komponenttesting.

ISO standard (ISO/IEC/IEEE 29119-1) har mer informasjon om konsepter for testing.

1.2 Hvorfor er testing nødvendig?

Grundig og nøyaktig testing av komponenter og systemer og deres tilhørende dokumentasjon kan hjelpe til å minske risikoen for at feil skjer i drift. Når feil blir funnet og deretter rettet, bidrar dette til komponentenes eller systemets kvalitet. I tillegg kan testing være påkrevd for å oppfylle kontraktsmessige krav, juridiske krav, industri- eller bransjestandarder.

1.2.1 Hvordan testing bidrar til suksess

Gjennom hele historien til databehandlingen har det vært vanlig at programvare og systemer blir levert til drift i en “umoden” tilstand og at de, fordi de inneholder feil, forårsaker problemer eller ikke møter interessentenes behov. Bruk av passende testteknikker kan redusere hyppigheten av slike problematiske leveranser. Forutsetningen er at disse teknikkene blir brukt med riktig testkompetanse, på de rette testnivåene og på de passende punktene i utviklingsmodellen. Eksempler er:

- La testere delta i reviews av krav eller i detaljering av brukshistorier kan føre til at de finner feil i disse arbeidsresultatene. Å identifisere og rette feil i krav minsker risikoen for å utvikle feil eller ikke testbar funksjonalitet
- La testere arbeide tett sammen med løsningsarkitekter mens systemet blir konstruert kan øke begge siders forståelse for designen og hvordan den kan testes. Denne økte forståelsen kan minske risikoen for fundamentale designfeil og gjøre det mulig å identifisere tester tidlig
- La testere arbeide tett sammen med utviklere mens koden blir laget kan øke begge siders forståelse av koden og hvordan den kan testes. Denne økte forståelsen kan minske risikoen for feil i koden og testene
- La testere verifisere og validere programvaren før leveranse kan finne feil som ellers kunne ha blitt oversett, og støtte prosessen med å rette feilene (dvs. debugging). Dette øker sannsynligheten for at programvaren oppfyller interessentenes behov og krav

I tillegg til disse eksempler bidrar oppfyllelse av definerte testmål (se avsnitt 1.1.1) til generell suksess med programvareutvikling og vedlikehold.

1.2.2 Kvalitetssikring og testing

Ofte brukes betegnelsen kvalitetssikring (eng: QA) når man omtaler testing. Man overser da at kvalitetssikring og testing ikke er det samme, selv om de henger sammen. De inngår i et mer omfattende konsept, kvalitetsstyring. Kvalitetsstyring omfatter alle aktiviteter for å lede og styre en organisasjon mht. kvalitet. Dette inkluderer blant annet aktivitetene kvalitetskontroll og kvalitetssikring.

Kvalitetssikring fokuserer typisk på å følge ordentlige prosesser, for å sørge for tillit til at det rette kvalitetsnivå blir oppnådd.

Når prosesser utføres ordentlig, er vanligvis arbeidsresultatene som er laget ved disse prosessene av høyere kvalitet. Dette bidrar til å forebygge feil. I tillegg er feilkildeanalyse viktig å bruke. Dermed finner en årsakene til feilene og kan rette disse. Det en finner ved feilkildeanalyse bør en bruke i retrospektiver for å forbedre prosessene. Alt dette er viktig for effektiv kvalitetssikring.

Kvalitetskontroll omfatter forskjellige aktiviteter, også testaktiviteter som hjelper med å oppnå de rette kvalitetsnivå. Testaktiviteter er del av den totale utviklings- eller vedlikeholdsprosessen for programvare.

Fordi kvalitetssikring handler om ordentlig utføring av hele prosessen, støtter kvalitetssikring ordentlig testing. Som beskrevet i avsnittene 1.1.1 og 1.2.1, bidrar testing til å oppnå kvalitet på mange måter.

1.2.3 Errors, Defects og Failures

Anmerkning: De engelske begrepene brukes her og i eksamen fordi det er vanskelig å oversette dem bokstavelig og kort til norsk!

En person kan gjøre en feil (error, mistake). Den kan lede til introduksjon av en feil (defect, fault, bug) i programkoden eller i et annet arbeidsresultat. Hvis en feil (defect) i koden blir utført, kan det lede til en (observerbar) feil eller feilsituasjon (failure).

Det gjelder ikke under alle omstendigheter. F.eks. krever noen feil (defects) meget spesielle inndata eller forutsetninger for å lede til feil eller feilsituasjoner (failure), som kan opptre sjelden eller aldri.

Feil (errors) kan skje pga. mange årsaker, som:

- Tidspress
- Det er menneskelig å feile
- Manglende erfaring og/eller kompetanse
- Misforståelser mellom deltakere i et prosjekt, bl.a. feilkommunikasjon om krav og design
- Kompleksitet av kode, design, arkitektur, det underliggende problemet som skal løses og/eller teknologien som blir brukt
- Misforståelser om kommunikasjon og grensesnitt i eller utenfor systemet, spesielt hvis det er mange grensesnitt
- Nye eller uvante teknikker eller teknologi

Feilsituasjoner (failures) kan oppstå pga. feil (defects) i koden. Betingelser i omgivelsen kan også forårsake feilsituasjoner (failures). F.eks. kan stråling, elektromagnetiske felter og forurensning sørge for trøbbel i firmware eller ha innflytelse på utføringen av programvaren ved å endre betingelsene selve hardwaren virker under.

Ikke alle uforventede testresultater er feil (failures). Såkalte falske positive (feilalarmer) kan opptre pga. feil i måten testene ble utført på eller pga. feil i testdata, testomgivelse eller annet testmateriell, eller av andre grunner. Det motsatte kan også skje, hvor liknende feil fører til falske negative. Falske negative er tester som ikke finner feil som de egentlig burde ha funnet. Falske positive blir rapportert som feil, men er i virkeligheten ingen feil.

1.2.4 Feil, feilkilder og virkninger av feil

Feilkildene er de tidligste aksjonene eller betingelsene som bidro til å lage feil. Feil kan bli analysert for å identifisere deres feilkilder, med det formål å minske forekomsten av liknende feil i framtiden. Når en

fokuserer på de mest signifikante feilkilder, kan feilkildeanalyse føre til prosessforbedringer som forebygger et vesentlig antall mulige framtidige feil.

F.eks. at feil renteberegning av banklån fører til kundeklager. Dette kan være forårsaket av bare en feil linje i programkoden. Den feile koden ble skrevet pga. en uklar brukshistorie, fordi produkteieren misforsto hvordan renter skal beregnes. Hvis en stor del av feilene ligger i renteberegningen og disse feilene har sin årsak i liknende misforståelser, kan produkteierne bli opplært i temaet renteberegning for å redusere denne type feil i framtiden.

I dette eksempel er kundeklagene følgen av feil. Feilbeløpene er “failures”. Den gale beregningen i koden er en “defect”. Den opprinnelige årsaken “defect” er en uklarhet i brukerhistorien. Feilkilden var mangel på kunnskap hos produkteieren, noe som førte til at produkteieren gjorde en feil «error» da han skrev brukerhistorien.

Prosessen for feilkildeanalyse blir diskutert i ISTQB-ETM Expert Level Testledelse Syllabus og ISTQB-EITP Expert Level Improving the Testprocess Syllabus.

1.3 De syv testprinsippene

En rekke testprinsipper ble foreslått i løpet av de siste 50 år. Disse er generelle retningslinjer for all testing.

1. Testing viser at feil er tilstede, ikke deres fravær

Testing kan vise at feil er tilstede i testobjektet, men kan ikke bevise at det ikke finnes feil. Testing reduserer sannsynligheten for at det finnes feil i programvaren som ikke er blitt oppdaget. Men selv om ingen feil blir funnet, er testing ingen bevis for korrekthet.

2. Fullstendig testing er umulig

Å teste alt (alle kombinasjoner av inputverdier og forutsetninger) er umulig, bortsett fra for trivielle tilfelle. Istedenfor å forsøke å teste fullstendig, bør en bruke risikoanalyse, testteknikker og prioritering for å fokusere testinnsatsen.

3. Tidlig testing sparer tid og penger

For å finne feil tidlig, bør en starte både statiske og dynamiske testaktiviteter så tidlig som mulig i programvareutviklingens livssyklus. Tidlig testing blir noen ganger betegnet som ”shift left”. Å teste tidlig i livssyklusen hjelper til å minske eller unngå kostbare endringer (se avsnitt 3.1).

4. Feil samler seg i klynger

Et mindre antall moduler inneholder vanligvis de fleste feil som finnes under testing før leveransen. Ellers kan et mindre antall moduler være ansvarlig for de fleste feil i drift. Forutsagte klynger av feil og de observerte klyngene under test og i drift er viktig input til risikoanalyse som brukes til å fokusere testinnsatsen (som det også blir nevnt i prinsipp 2)

5. Pass på pesticid-paradokset!

Dersom en gjentar de samme testene gang på gang, vil disse testene etter en stund ikke finne flere feil. Systemet blir på en måte immun mot disse testene. For å finne nye feil, kan det være påkrevd å endre eksisterende tester og testdata, og det kan være behov for nye tester. Testene er ikke lenger effektive til å finne feil, akkurat som pesticider mister sin effekt. F.eks. finner automatiserte regresjonstester ofte få feil pga. pesticid-paradokset.

6. Testing er avhengig av konteksten

Ulik kontekst krever ulik test. F.eks. blir sikkerhetskritisk programvare testet annerledes enn en app for mobiler. Som et annet eksempel er test i et smidig prosjekt annerledes enn test i et prosjekt med sekvensiell livssyklus (se avsnitt 2.1).

7. Feilslutning vedrørende fravær av feil

Noen organisasjoner forventer at testere kan utføre alle mulige tester og finner alle mulige feil. Men prinsippene 1 og 2 viser oss at dette ikke er mulig. I tillegg er det feil å tro at bare det å finne og rette et stort antall feil vil sikre at systemet blir en suksess. F.eks. når en tester grundig alle spesifiserte krav og retter feilene kan en fortsatt ha et system som er vanskelig å bruke, som ikke oppfyller brukernes behov og forventninger, eller som er dårligere enn andre konkurrerende systemer.

Se Myers 2011, Kaner 2002, og Weinberg 2008 for eksempler av disse og andre testprinsipper.

1.4 Testprosessen

Det finnes ingen universell testprosess for programvare. Men det finnes allment gyldige testaktiviteter som er grunnlag for at testing oppnår suksess. Disse testaktiviteter til sammen er en testprosess. I hver gitt situasjon avhenger den rette, spesifikke testprosessen av en rekke faktorer. Det kan diskuteres i en organisasjons teststrategi hvilke testaktiviteter som inngår i denne testprosessen, hvordan disse aktivitetene gjøres og når de skjer.

1.4.1 Testprosessen og kontekst

Faktorer i konteksten som har innflytelse på testprosessen for en organisasjon omfatter, men er ikke begrenset til:

- Utviklingsmodellen for programvare og prosjektets metoder
- Testnivåer og testtyper som vurderes
- Produkt- og prosjektrisikoer
- Forretningsområde
- Restriksjoner, inkludert disse og andre:
  - Budsjetter og ressurser
  - Tidsrammer
  - Kompleksitet
  - Kontraktuelle krav
  - Myndighetskrav og regler
- Organisasjonens policy og praksis
- Påkrevde interne og eksterne standarder

De følgende avsnittene beskriver generelle aspekter av organisasjonens testprosesser ved å nevne følgende:

- Testaktiviteter og oppgaver
- Arbeidsresultater fra testingen
- Sporbarhet mellom testgrunnlag og testingens arbeidsresultater

Det er veldig nyttig hvis testgrunnlaget inneholder målbare dekningskriterier. Dette gjelder for hvert testnivå og testtype. Dekningskriteriene kan være gode nøkkelindikatorer for å drive aktivitetene som demonstrerer at testmål blir nådd (se avsnitt 1.1.1).

F.eks. kan testgrunnlaget for en mobil applikasjon omfatte en kravliste og en liste med mobile enheter som skal støttes. Hvert krav er et element av testgrunnlaget. Hver mobil enhet som støttes er også et element av testgrunnlaget. Dekningskriteriene kan kreve minst et testtilfelle for hvert element av testgrunnlaget.

ISO standard ISO/IEC/IEEE 29119-2 har mer informasjon om testprosesser.

1.4.2 Testaktiviteter og oppgaver

En testprosess består av følgende hovedgrupper av aktiviteter:

- Testplanlegging
- Testovervåking og -styring
- Testanalysen
- Testdesign
- Testimplementering
- Testutføring
- Testavslutning

Hver gruppe aktiviteter består av enkeltaktiviteter som beskrives i avsnittene under. Hver aktivitet i en gruppe kan igjen bestå av flere individuelle oppgaver som kan variere fra et prosjekt eller utgivelse til et annet.

Selv om mange av disse aktivitetsgruppene ser logisk sekvensielle ut, blir de ofte iterative i praksis. F.eks. bruker smidig utvikling små iterasjoner av design, bygging og test som skjer kontinuerlig. De støttes av kontinuerlig planlegging. Dermed skjer testaktiviteter også kontinuerlig og iterativt ved bruk av denne metoden. Selv i sekvensiell utvikling vil den stegvise logiske rekkefølgen av aktiviteter kreve overlapp, kombinasjon, samtidighet eller utelatelse. Altså kreves vanligvis at disse hovedaktivitetene tilpasses konteksten til prosjektet og systemet.

Testplanlegging

Testplanlegging definerer testmål og måten disse skal oppnås på, samtidig som en følger restriksjonene i konteksten. F.eks. å spesifisere passende testteknikker og oppgaver og å lage en kjøreplan for å møte et tidsmål. Testplaner kan endres basert på tilbakemelding fra aktivitetene for overvåning og styring. Testplanlegging blir mer forklart i avsnitt 5.2.

Testovervåking og -styring

Testovervåking er den kontinuerlige sammenligningen av virkelig fremdrift med testplanen ved å bruke måledata for testoppfølging som er definert i testplanen. Teststyring betyr at en gjør nødvendige tiltak for å oppfylle målene i testplanen. Disse kan bli oppdatert over tid.

Testovervåking og -styring understøttes av at en evaluerer sluttkriteriene. I smidig utvikling kan de inngår i "definition of done" (se ISTQB-AT Foundation Level Agile Tester Extension Syllabus). F.eks. kan evalueringen av sluttkriterier for testutføring som del av et gitt testnivå omfatte:

- Sjekke testresultater og logger mot spesifiserte dekningskriterier
- Bedømme nivået på komponent- eller systemkvalitet, basert på testresultater og logger
- Bestemme om det behøves flere tester. F.eks. hvis testene som opprinnelig var ment til å oppnå et visst dekningsnivå av produktrisikoer ikke gjorde det, slik at flere tester må skrives og utføres

Testframdrift i forhold til planen presenteres for interessentene i framdriftsrapport for test. Disse omfatter også feil fra planen og informasjon til støtte for eventuelt å stoppe testingen.

Testovervåking og -styring blir videre forklart i avsnitt 5.3.

Testanalyse

Under testanalyse gjennomgås testgrunnlaget for å finne funksjoner som kan testes og for å definere tilhørende testbetingelser. Med andre ord: testanalysen bestemmer "hva som skal testes". Dette bør skje i form av målbare dekningskriterier.

Testanalyse består av følgende hovedaktiviteter:

- Analysere testgrunnlaget (for komponenten eller systemet) passende til testnivået, f.eks.:
    - Kravspesifikasjoner, som forretningskrav, funksjonskrav, systemkrav, brukerhistorier, "episke brukerhistorier", brukstilfeller, eller liknende arbeidsresultater som f.eks. spesifiserer ønsket funksjonell og ikke-funksjonell oppførsel av en komponent eller et system
    - Informasjon om design og implementasjon, som system- eller programvare-arkitekturdiagrammer eller -dokumenter, designspesifikasjoner, flyt av kall, modelldiagrammer (f.eks. UML), grensesnittspesifikasjoner, eller liknende arbeidsresultater som spesifiserer testobjektets struktur
    - Implementasjonen av komponenten eller systemet, inklusive kode, beskrivelser av database, kall og grensesnitt
    - Rapporter fra risikoanalyse som kan omfatte funksjonelle, ikke-funksjonelle og/eller strukturelle aspekter
- Evaluere testgrunnlaget og testobjektet for å finne ulike typer feil, så som:
    - Tvetydigheter
    - Utelatte ting
    - Uoverensstemmelser
    - Unøyaktigheter
    - Motsetninger
    - Overflødige ting

- Identifisere funksjoner som skal testes
- Definere og prioritere testbetingelser for hver funksjon basert på analyse av testgrunnlaget. Dette gjøres ved å ta i betraktning funksjonelle, ikke-funksjonelle og strukturelle egenskaper, andre tekniske og forretningsmessige faktorer og risikonivåer
- Sørge for sporbarhet begge veier mellom hvert element av testgrunnlaget og de tilhørende testbetingelser (se avsnitt 1.4.3 og 1.4.4)

Svartboks, hvitboks og erfaringsbaserte testteknikker er nyttige i testanalysen (se kapittel 4) for å minske sannsynligheten for å utelate viktige testbetingelser og for å definere mer nøyaktige og presise testbetingelser.

I testanalysen finner en testbetingelser som kan brukes som testmål. Dette er typiske arbeidsresultater i noen typer av erfaringsbasert testing (se avsnitt 4.4.2). Hvis disse testmål er sporbare til testgrunnlaget, kan en måle den oppnådde testdekningen.

Å finne feil under testanalysen er en viktig mulig nytteeffekt, spesielt hvor ingen annen review brukes og/eller der testanalysen er del av review. En slik testanalyse verifiserer ikke bare om kravene er konsistent, ordentlig uttrykt og fullstendig. De validerer også om kravene uttrykker behov fra kunde, bruker og andre interessenter rett. Det finnes teknikker som bidrar til å verifisere, validere og finne feil i brukerhistorier og akseptansekriterier (se ISTQB Foundation Level Agile Tester Extension syllabus).

Testdesign

Under testdesign blir testbetingelsene utarbeidet til overordnete testtilfelle, samlinger av overordnete testtilfelle, og annet testmateriell. Altså: testanalyse svarer på spørsmålet “HVA skal en teste?” mens testdesign svarer på spørsmålet “HVORDAN teste?”.

Testdesign omfatter følgende hovedaktiviteter:

- Konstruere og prioritere testtilfelle og samlinger av testtilfelle
- Identifisere nødvendige testdata for å understøtte testbetingelser og testtilfelle
- Konstruere testmiljøer og identifisere nødvendig infrastruktur og verktøy
- Opprette toveis sporbarhet mellom testgrunnlaget, testbetingelser, testtilfeller og testprosedyrer (se avsnitt 1.4.4)

Å videreutvikle testbetingelser til testtilfelle og samlinger av testtilfelle under testdesign innebærer ofte at en bruker testteknikker (se kapitel 4).

Som testanalyse kan testdesign også føre til at en identifiserer liknende feil i testgrunnlaget. Dette er en viktig potensiell nytteeffekt.

Testimplementering

Under testimplementeringen lages og/eller ferdigstilles det nødvendige testmateriell for testutføringen, inklusive å sette sammen testtilfellene til testprosedyrer. Testimplementeringen svarer på spørsmålet “Har vi nå alt på plass for å kjøre testene?”.

Testimplementering omfatter følgende hovedaktiviteter:

- Utvikle og prioritere testprosedyrer, og (hvis relevant) lage automatiserte testscripter
- Lage testsuiter fra testprosedyrene og (hvis en har) automatiserte testscripter
- Arrangere testsuitene i en kjøreplan slik at testutføringen blir effektiv (se avsnitt 5.2.4)

- Sette opp et testmiljø (som kan omfatte testrammer, virtualisering av tjenester, simulatorer og annen infrastruktur) og å verifisere at alt som trengs er satt opp korrekt
- Forberede testdata og sikre at de er korrekt lastet opp i testmiljøet. Husk beskyttelse av persondata
- Verifisere og oppdatere sporbarhet i begge retninger mellom testgrunnlaget, testbetingelser, testtilfelle, testprosedyrer og testsuiter (se avsnitt 1.4.4)

Oppgavene for testdesign og testimplementering blir ofte kombinert.

I utforskende testing og andre typer erfaringsbasert testing kan testdesign og -implementering utføres og bli dokumentert som del av testutføringen. Utforskende testing kan bli basert på testcharter (som lages som del av testanalysen). Utforskende tester blir utført umiddelbart når de er konstruert og implementert (se avsnitt 4.4.2).

Testutføring

Under testutføring blir testsuiter utført i samsvar med kjøreplanen.

Testutføring omfatter følgende hovedaktiviteter:

- Loggføre ID og versjoner av testobjekten(e), testverktøy og testmateriell
- Utføre tester enten manuelt eller ved å bruke testutføringsverktøy
- Sammenligne aktuelle resultater med de forventede resultatene
- Analysere hendelser og feil for å finne deres sannsynlige årsaker. F.eks. kan feilsituasjoner opptre pga. kodefeil, men det kan også være falske positive (se avsnitt 1.2.3)
- Rapportere feil basert på feilene som observeres (se avsnitt 5.6)
- Logge resultatene fra testutføringen (f.eks. OK, feil, blokkert)
- Gjenta testaktiviteter enten fordi en har endret/rettet noe etter en feil, eller som del av planlagt testing (f.eks. å utføre en rettet test, retesting og/eller regresjonstesting)
- Verifisere og oppdatere sporbarhet i begge retninger mellom testgrunnlaget, testbetingelser, testtilfeller, testprosedyrer og testresultater

Testavslutning

Her samles data fra ferdige testaktiviteter for å ta være på erfaringer, testmateriell og annen relevant informasjon. Testavslutningsaktiviteter skjer ved prosjektets milepæler. F.eks. når et programvaresystem blir levert, et testprosjekt er ferdig eller avbrutt, en iterasjon i et smidig prosjekt er avsluttet, et testnivå er fullført eller en vedlikeholdsleveranse er ferdig.

Testavslutning omfatter følgende hovedaktiviteter:

- Verifisere om alle rapporterte feil er lukket
- Lage endringsønsker og “product backlog items” for «failures» som ikke ble klassifisert som «defects» ved slutten av testutføringen
- Lage en sluttrapport for test for interessentene
- Avslutte og arkivere testmiljøet, testdata, testinfrastruktur og annet testmateriell for senere gjenbruk
- Overlevere testmateriellet til vedlikehold/forvaltning, andre prosjektgrupper og/eller andre interessenter som kan ha nytte av det

- Analysere erfaringer fra de fullførte testaktivitetene for å bestemme hva slags endringer en må gjøre i fremtiden (for iterasjoner, leveranser og prosjekter)
- Bruke informasjonen som er samlet for å forbedre testprosessen

1.4.3 Arbeidsprodukter/-resultater fra testingen

I denne oversettelsen bruker vi termene arbeidsprodukter og arbeidsresultater (eng: «work products») i samme betydning.

I testprosessen blir det produsert arbeidsprodukter. Akkurat som det er en signifikant variasjon i måten som organisasjoner implementerer testprosessen, er der også en signifikant variasjon i arbeidsresultatene som lages under testprosessen. Dette gjelder måtene disse arbeidsresultatene blir organisert og styrt, og hva man kaller dem. Dette pensum holder seg til testprosessen som er beskrevet ovenfor og til arbeidsresultatene som er beskrevet i dette pensum og i ISTQB Terminologilisten.

ISO standard (ISO/IEC/IEEE 29119-3) kan også tjene som en retningslinje for arbeidsresultater.

Mange av disse arbeidsresultatene som er beskrevet i dette avsnitt kan også bli generert og styrt ved hjelp av verktøy for teststyring og feilhåndtering (se kapittel 6).

Arbeidsresultater fra testplanlegging

Disse omfatter vanligvis en eller flere testplaner. Testplanen inneholder informasjon om testgrunnlaget, som de andre testarbeidsresultatene vil bli knyttet til ved hjelp av informasjon om sporbarhet (se lenger nede og avsnitt 1.4.4). Den inneholder også sluttkriterier som brukes under testovervåking og -styring. Testplaner er beskrevet i avsnitt 5.2.

Arbeidsresultater fra testovervåking og -styring

Disse omfatter vanligvis ulike typer av testrapporter, f.eks. framdriftsrapporter (som produseres fortløpende eller regelmessig) og sluttrapport for tester (som produseres ved forskjellige milepæler). Alle testrapporter bør gi relevante detaljer om testframdriften ved rapporteringstidspunktet, inklusive en oppsummering av resultatene fra testutføringen så snart disse blir tilgjengelige.

Arbeidsresultatene fra testovervåking og -styring bør også inneholde informasjon om det som prosjektledelsen er opptatt av, som ferdigstilling av oppgavene, ressurstildeling og -bruk og innsats.

Testovervåking og -styring, og arbeidsresultatene som lages ved disse aktivitetene er videre forklart i avsnitt 5.3.

Arbeidsresultater fra testanalyse

Disse omfatter definerte og prioriterte testbetingelser. Ideelt vil disse være sporbare begge veier til de spesifikke elementene i testgrunnlaget de dekker. For utforskende testing kan testanalysen omfatte at en spesifiserer testcharter. Testanalyse kan også resultere i at feil i testgrunnlag blir oppdaget og rapportert.

Arbeidsresultater fra testdesign

Testdesign resulterer i testtilfelle og samlinger av testtilfelle for å utføre testbetingelsene som er definert i testanalysen. Det er ofte en god vane å lage overordnete testtilfelle, uten konkrete verdier for inndata og forventede resultater. Slike overordnete testtilfelle kan gjenbrukes i flere testsykluser med ulike konkrete dataverdier. Ideelt vil det være toveis sporbarhet mellom testtilfelle og testbetingelsen(e) den dekker.

Testdesign resulterer også i utarbeidelse og/eller identifikasjon av de nødvendige testdata, konstruksjon av testmiljøet og identifikasjon av nødvendig infrastruktur og verktøy. Men omfanget for dokumentering av alt dette varierer vesentlig.

Testbetingelsene som er definert i testanalysen kan under testdesign bli utarbeidet i mer detalj.

Arbeidsresultater fra testimplementeringen

Disse omfatter:

- Testprosedyrer
- Testsuiter
- En kjøreplan

Ideelt kan en demonstrere at dekningskriteriene fra testplanen er oppfylt. Dette gjøres ved hjelp av toveis sporbarhet mellom testprosedyrene og de spesifikke elementene i testgrunnlaget, gjennom testtilfelle og testbetingelser.

I noen tilfeller involverer testimplementering at en lager arbeidsresultater ved å bruke verktøy eller til bruk for verktøy. F.eks. virtualisering av tjenester og automatiserte testscripter.

Testimplementering kan også lage og verifisere testdata og testmiljø. Fullstendigheten for dokumentasjon av testdata og/eller verifisering av testmiljø kan variere vesentlig.

Testdata skal gi konkrete verdier til inputs og forventede resultater av testtilfellene. Slike konkrete verdier sammen med eksplisitte anvisninger om bruken av disse konkrete verdiene endrer overordnete testtilfelle til utførbare detaljerte testtilfelle. De samme overordnete testtilfelle kan bruke ulike testdata når de blir utført på ulike leveranse av testobjektet. De konkrete forventede resultatene som er forbundet med konkrete testdata blir identifisert ved hjelp av et testorakel.

I utforskende testing kan noen arbeidsresultater fra testdesign og -implementering bli laget under testutføringen. Det kan variere vesentlig i hvilken grad utforskende tester og deres sporbarhet til spesifikke elementer i testgrunnlaget blir dokumentert.

Testbetingelsene som er definert i testanalysen kan videre detaljeres under testimplementeringen.

Arbeidsresultater fra testutføringen

Disse omfatter:

- Dokumentasjon av status for de individuelle testtilfelle eller testprosedyrer f.eks. klar til utføring, OK/godkjent, feilet, blokkert, tatt bort med vilje, etc.
- Feilrapporter (se avsnitt 5.6)
- Dokumentasjon om hvilke testelementer, testobjekter, testverktøy og testmateriell ble brukt i testingen

Ved slutten av testutføringen kan i beste fall status for hvert element i testgrunnlaget bestemmes og rapporteres gjennom sporbarhet med de(n) tilhørende testprosedyren(e). F.eks. kan en rapportere for hvilke krav alle testene er godkjent, hvilke krav som har feilede tester og for hvilke krav det fortsatt finnes planlagte og ikke utførte tester. Dette muliggjør verifisering av at dekningskriteriene har blitt oppfylt. Dermed kan en rapportere testresultater i en form som er forståelig for interessentene.

Arbeidsresultater fra testavslutningen

Disse omfatter sluttrapport for tester, aksjonspunkter for forbedring av framtidige prosjekter eller iterasjoner (f.eks. etter en retrospektive), endringsønsker eller «product backlog items» og ferdigstilt testmateriell.

1.4.4 Sporbarhet mellom testgrunnlaget og arbeidsresultatene fra testen

Det er viktig å etablere og vedlikeholde sporbarhet gjennom hele testprosessen mellom hvert element i testgrunnlaget og de ulike arbeidsresultatene som hører til dette elementet, som beskrevet før. Dette er

viktig for å få effektiv testovervåking og -styring. I tillegg til evaluering av testdekning, understøtter god sporbarhet dette:

- Analysere følgene av endringer
- Gjøre testing reviderbart
- Tilfredsstille kriteriene for organisasjons IKT-styringsmodell
- Gjøre framdriftsrapporter og sluttrapport for test mer forståelig ved å ta med status for elementer i testgrunnlaget, f.eks. krav som ble godkjent eller ikke i sine tester og krav som har tester som fortsatt venter på å bli utført
- Beskrive de tekniske aspektene av testingen for interessenter på en forståelig måte
- Gi informasjon for å kunne bedømme produktkvalitet, prosessevne og prosjektframdrift mot forretningsmål

Noen testledelsesverktøy gir modeller for testarbeidsresultater som samsvarer med noen eller alle test-arbeidsresultater som ble beskrevet i dette avsnittet. Noen organisasjoner bygger sine egne ledelsessystemer for å organisere arbeidsresultatene og gi informasjon om sporbarheten de krever.

1.5 Testingens psykologi

Programvareutvikling og -testing gjøres av mennesker. Derfor har menneskelig psykologi en viktig innflytelse.

1.5.1 Menneskelig psykologi og testing

Å identifisere feil under statisk test som review av krav (eller i et møte der en utarbeider brukshistorier) eller å identifisere feil under dynamisk testutføring kan oppfattes som kritikk av produktet eller dets forfatter. Et element av menneskelig psykologi kalt «confirmation bias» kan gjøre det vanskelig å godta informasjon som er i konflikt med de oppfatninger en har fra før. F.eks. har utviklere denne «confirmation bias» som gjør det vanskelig å godta at koden inneholder feil. Dette fordi utviklere forventer at deres kode er rett.

I tillegg til dette finnes andre kognitive skjevheter som gjør det vanskelig for folk å forstå eller akseptere informasjon som testingen produserer. I tillegg er det et menneskelig trekk å anklage den som kommer med dårlige nyheter, og informasjon fra testingen inneholder ofte dårlige nyheter.

Som et resultat av disse psykologiske faktorene, kan noen oppfatte testing som en destruktiv aktivitet, selv om testing i stor grad bidrar til at prosjekter går framover og til produktkvalitet (se avsnittene 1.1 og 1.2). For å redusere disse negative oppfatninger, bør informasjon om feil kommuniseres konstruktivt og saklig. Slik kan en redusere spenninger mellom testere og analytikere, produkteiere, konstruktører og utviklere. Dette gjelder for både statisk og dynamisk testing.

Testere og testledere må ha gode evner til mellommenneskelig kommunikasjon for å kunne diskutere feil, testresultater, testframdrift og risiko effektivt, og for å bygge opp positive relasjoner med kollegene. Måter å kommunisere bra omfatter de følgende eksempler:

- Start med å samarbeide heller enn å kjempe mot hverandre. Minn alle om det felles mål at systemene får bedre kvalitet
- Understrek testingens nytteverdi. F.eks. kan informasjon om feil hjelpe forfatterne å forbedre sine arbeidsresultater og sine evner. For en organisasjon kan feil funnet og rettet under testing spare tid og penger og redusere den totale risikoen med produktkvaliteten

- Kommuniser testresultater og andre funn på en nøytral, faktabasert måte uten å kritisere personen som laget det som er feil. Skriv objektive og saklige feilrapporter og anmerkninger fra reviews
- Forsøk å forstå hvordan den andre personen føler og grunnene hvorfor de eventuelt reagerer negativt på informasjonen
- Få bekreftet at den andre personen har forstått hva som har blitt sagt – og omvendt

Typiske mål for testen ble nevnt før (se avsnitt 1.1). En klar definisjon av de rette testmål har viktige psykologiske følger. De fleste mennesker tenderer til å justere sine planer og oppførsel med målsetningene satt av teamet, ledelsen eller andre interessenter. Det er også viktig at testere holder seg til disse målene med minimal personlig forutinntatthet.

1.5.2 Tankesettet til testere og utviklere

Utviklere og testere tenker ofte forskjellig. Utviklernes viktigste mål er å konstruere og bygge et produkt. Som diskutert før, inkluderer målene til testingen å verifisere og validere produktet, å finne feil før leveranse osv. Dette er ulike mål som krever ulike tankesett. Når en bringer disse tankesett sammen, bidrar dette til å oppnå et høyere nivå av produktkvalitet.

Et tankesett gjenspeiler et individs antagelser og foretrukne metoder for å gjøre beslutninger og løse problemer.

Tankesettet til en tester bør omfatte nysgjerrighet, profesjonell pessimisme, et kritisk øye, oppmerksomhet på detaljer, og en motivasjon for god og positiv kommunikasjon og relasjoner. Tankesettet til en tester vokser og modner når testeren får mer erfaring.

En utviklers tankesett kan omfatte noen av elementene av en testers tankesett, men suksessfulle utviklere er ofte mer interessert i å konstruere og bygge løsninger enn i å tenke over hva som kunne være feil med disse løsningene. I tillegg gjør «confirmation bias» det vanskelig å fine feil i deres eget arbeide.

Med det rette tankesettet kan utviklere teste sin egen kode. Ulike utviklingsmodeller for programvare har ofte forskjellige måter å organisere testerne og testaktiviteter. Å få noen testaktiviteter utført av uavhengige testere øker effektiviteten i å finne feil. Dette er særlig viktig ved store, komplekse eller sikkerhetskritiske systemer. Uavhengige testere har med seg et perspektiv som er ulik perspektivet til forfatterne av arbeidsresultatene (f.eks. analytikere, produkteiere og programmerere), fordi de har andre «cognitive bias» enn forfatterne.

2 Testing i programvarens livssyklus — 100 minutter

Nøkkelord

akseptansetesting, akseptansetesting av samsvar med kontrakt, akseptansetesting av samsvar med lover og regler, alfatesting, betatesting, brukerakseptansetest, driftsakseptansetesting, funksjonell testing, hvitboks testing, hyllevare (COTS), ikke-funksjonell testing, integrasjonstesting, komponentintegrasjonstesting, komponenttesting, konsekvensanalyse, regresjonstesting, retesting, sekvensiell utviklingsmodell, systemintegrasjonstesting, systemtesting, testgrunnlag, testmiljø, testmål, testnivå, testobjekt, testtilfelle, testtype, vedlikeholdstesting

OBS. I dette kapitler er i noen tilfeller ordet «programvare» fjernet for bedre lesbarhet.

OBS. Metodikker eller prosesser kan bli beskrevet i modeller. Vi opplever at i dette kapitlet ordene modeller og metodikker/prosesser ikke er brukt presis eller om og om.

Læremål

2.1 Livssyklusmodeller

FL-2.1.1 (K2) Forklar relasjonene mellom utviklingsaktiviteter og testaktiviteter

FL-2.1.2 (K1) Identifiser grunnlaget for hvorfor livssyklusmodeller må tilpasses til prosjektets kontekst og produktegenskaper

2.2 Testnivåer

FL-2.2.1 (K2) Sammenlign de forskjellige testnivåene sett fra testformål, testgrunnlag, testobjekter, typiske feil, og testtilnærming og ansvarsfordeling

2.3 Testtyper

FL-2.3.1 (K2) Sammenlign funksjonell, ikke-funksjonell og hvitboks testing

FL-2.3.2 (K1) Gjenkjenn at funksjonell, ikke-funksjonell og hvitboks testing kan gjennomføres i alle testnivåer

FL-2.3.3 (K2) Sammenlign formålene med retesting og regresjonstesting

2.4 Vedlikehold Testing

FL-2.4.1 (K2) Oppsummer årsaker for vedlikeholdstesting

FL-2.4.2 (K2) Beskriv rollen av konsekvensanalyse ved vedlikeholdstesting

2.1 Livssyklusmodeller

En livssyklusmodell beskriver aktivitetene for alle faser i et utviklingsprosjekt, og hvordan aktivitetene er relatert til og avhengig fra hverandre. Det er flere livssyklusmodeller som alle krever en tilpasset testtilnærming.

2.1.1 Utvikling og testing

Det er viktig at en tester kjenner til de mest brukte livssyklusmodeller slik at hun kan spesifisere de riktige og mest fornuftige testaktiviteter for den aktuelle modellen.

Det er en del testkarakteristikker som er felles for alle livssyklusmodeller:

- Hver utviklingsaktivitet har en tilsvarende testaktivitet
- Hvert testnivå har et spesifikt testformål
- Testanalyse og testdesign for hvert testnivå bør starte mens den tilsvarende utviklingsaktiviteten pågår

Testere bør delta i diskusjoner for spesifisering og forbedring av krav og design. De bør også delta i gransking av arbeidsprodukter, f.eks. kravspesifikasjon, design, brukerhistorier så snart utkast av disse foreligger.

Uansett hvilken utviklingsmodell som blir brukt, skal testplanlegging og -forberedelse begynne så tidlig som mulig. Dette er i samsvar med prinsippet om tidlig test.

Dette pensum er basert på to kategorier utviklingsmodeller:

- Sekvensielle utviklingsmodeller
- Trinnvise og iterative utviklingsmodeller

Sekvensielle utviklingsmodeller

Sekvensielle utviklingsmodeller beskriver utviklingsprosesser lineært, som en kjede av påfølgende aktiviteter. Den er basert på at enhver fase i prosessen kun skal starte når forgående fasen er avsluttet. I teorien skal det ikke være overlapp av faser, men i praksis ser man ofte at neste fase begynner allerede når pågående fase er under arbeid. Slik får en muligheter til å forbedre resultater til den pågående fasen.

Et kjent eksempel er fossefallsmodellen hvor utviklingsaktivitetene kravanalyse, design, koding og test blir gjennomført etter hverandre. I denne modellen blir testaktiviteter gjennomført etter at alle andre utviklingsaktiviteter er ferdig.

V-modellen er et annet eksempel på en sekvensiell modell, men her er testaktivitetene tett integrert med utviklingsaktivitetene og basert på prinsippet om tidlig test. I tillegg kobler modellen testnivåer til tilsvarende utviklingsfaser som gir ytterligere mulighet for tidlig forberedelse av test (se også avsnitt 2.2 om de ulike testnivåer). Utføring av hvert testnivå i denne modellen skjer etter hverandre, men det kan også være noe overlapp.

Sekvensielle utviklingsmodeller leverer typisk programvare som inkluderer alle spesifiserte funksjoner og funksjonaliteter, men det brukes ofte lengre perioder (måneder – år) før prosjektet kan avsluttes.

Trinnvise og iterative utviklingsmodeller

Med trinnvis og iterativ utvikling blir krav, design, kode og test realisert i mindre deler eller leveranser som resulterer i at den totale funksjonaliteten blir utvidet trinnvis. Størrelsen på disse delene kan variere hvor noen modeller produserer større delleveranser og andre modeller mindre store. Det kan f.eks. være en mindre feilretting av brukergrensesnitt eller ny søkefunksjon som blir levert.

Typisk for trinnvis og iterativ utvikling er at deler av funksjoner og funksjonaliteter blir spesifisert, designet, bygget og testet sammen i en serie av korte perioder/sykluser som ofte har en fast lengde. Påfølgende iterasjoner kan gjerne omfatte forbedringer eller endringer av tidligere levert funksjonalitet samt utvidelser av funksjonaliteten. Hver iterasjon skal levere fungerende programvare, som så blir utvidet i de neste iterasjonene til hele løsningen er levert eller utviklingen stoppes. Eksempler er:

- Rational Unified Process (RUP): Hver iterasjon er relativt lang (f.eks. to til tre måneder), og omfang av leveransene er tilsvarende store, f.eks. to eller tre sammensatte grupper av relaterte funksjoner
- Scrum: Hver iterasjon er relativt kort (f.eks timer, dager eller noen få uker), og omfang av leveransene er tilsvarende små, f.eks. noen forbedringer og/eller to eller tre nye funksjoner
- Kanban: Brukt med eller uten iterasjoner med fast lengde, som kan levere enten en enkel forbedring eller funksjon ved ferdigstillelse, eller kan gruppere noen funksjoner sammen for samtidig ferdigstillelse
- Spiral (eller prototyping): Basert på at det utvikles forslag for ny funksjonalitet, hvorav noen kan bli komplett omarbeidet eller til og med forlatt i etterfølgende iterasjoner

Komponenter eller systemer utviklet med bruk av disse metodene involverer ofte overlappende og gjentagende testnivåer gjennom hele utviklingen. Ideelt sett testes hver funksjon på flere testnivåer for hver iterasjon ettersom den beveger seg mot levering. I noen tilfeller bruker teamene kontinuerlig integrasjon eller kontinuerlig leveranse, som begge innebærer betydelig testautomatisering på flere testnivåer som en del av leveranseprosessen. I utviklingsarbeid der disse metodene blir brukt, bruker man ofte også selvorganiserende teams. Dette kan forandre måten testingen blir organisert på, samt relasjonen mellom testere og utviklere i teamet.

Disse metoder bygger et voksende system, som leveres til sluttbrukere funksjon for funksjon, iterasjon for iterasjon, eller mer tradisjonelt i en stor leveranse til slutt. Uansett om trinnet blir levert til sluttbrukere, er regresjonstesting stadig viktigere når systemet blir større.

I motsetning til sekvensielle modeller, kan iterative og trinnvise modeller levere brukbar programvare etter uker eller dager. Skal leveransen inkludere alle krav, kan en periode vare i noen måneder eller til og med år.

For mer informasjon om testing i sammenheng med smidig utvikling, se ISTQB-AT Foundation Level Agile Tester Extension Syllabus, Black 2017, Crispin 2008 og Gregory 2015.

2.1.2 Livssyklusmodeller i kontekst

Utviklingsmodeller må velges og tilpasses til prosjektet og produktegenskapene. En hensiktsmessig utviklingsmodell bør velges og tilpasses basert på prosjektmål, produkttype som utvikles, forretningsprioriteter (f.eks. tid til marked) og identifiserte produkt- og prosjektrisikoer. F.eks. utvikling og testing av et mindre internt administrativt system vil avvike fra utvikling og testing av et sikkerhetskritisk system, som f.eks. bilens bremsestyringssystem. Som et annet eksempel kan organisatoriske og kulturelle utfordringer i noen tilfeller hindre kommunikasjon mellom medlemmene i team, noe som kan hindre trinnvis utvikling.

Avhengig av prosjektets kontekst, kan det være nødvendig å kombinere eller reorganisere testnivåer og/eller testaktiviteter. F.eks. for integrering av et hyllevareprodukt med et større system, kan kunden utføre testing av interoperabilitet på systemintegrasjonstestnivå og på akseptansetestnivå (funksjonell og ikke-funksjonell, sammen med brukerakseptansetest og driftsakseptansetesting). Se avsnitt 2.2 Testnivåer og avsnitt 2.3 Testtyper.

I tillegg er det mulig å kombinere ulike utviklingsmodeller. F.eks. kan en V-modell brukes til utvikling og testing av «backend-systemer» og deres integrasjoner, mens en trinnvis utviklingsmodell kan brukes til utvikling og testing av brukergrensesnittet (UI) og dens funksjonaliteter. Prototyping kan gjerne brukes tidlig i prosjektet, og realiseres med hjelp av en trinnvis utviklingsmodell..

Tingenes internett (IoT), som kan bestå av mange forskjellige objekter, f.eks. enheter, produkter og tjenester, bruker vanligvis ulike utviklingsmodeller for hvert objekt. Dette gir en spesiell utfordring for utvikling av ulike IoT-versjoner. I tillegg legger livssyklus til slike objekter mer vekt på senere faser i utviklingens livssyklus etter at de har blitt tatt i operativ bruk, f.eks. drifts-, forvaltnings-, videreutviklings- og avinstalleringsfaser.

2.2 Testnivåer

Et testnivå er en gruppering av testaktiviteter som er organisert og administrert sammen. Hvert testnivå er en del av testprosessen, bestående av aktivitetene som beskrevet i avsnitt 1.4, utført avhengig av på hvilket utviklingsnivå programvaren befinner seg, fra individuelle enheter/komponenter til komplette systemer eller systemer av systemer. Testnivåer er relatert til andre aktiviteter innen utviklingsmodellen. Testnivåene som brukes i dette pensumet er:

- Komponenttesting
- Integrasjonstesting
- Systemtesting
- Akseptansetesting

De ulike testnivåene har følgende spesifikke egenskaper:

- Spesifikt mål eller hensikt
- Eget testgrunnlag for å avlede testtilfeller
- Testobjekt (dvs. hva skal bli testet)
- Typiske feil og feilsituasjoner som skal finnes
- Spesifikke tilnærminger og ansvarsfordeling

Hvert testnivå trenger et egnet testmiljø. For akseptansetesting er et produksjonslikt testmiljø ideelt, mens utviklere gjerne bruker sine egne utviklingsmiljøer for komponenttesting.

2.2.1 Komponenttesting

Mål med komponenttesting

Komponenttesting (også kjent som enhets- eller modultesting) fokuserer på komponenter som kan testes individuelt. Målene med komponenttesting er:

- Redusere risiko
- Verifisere om de funksjonelle og ikke-funksjonelle egenskapene er utviklet som spesifisert og designet
- Bygge tillit til komponentens kvalitet
- Finne feil i komponenten
- Forebygge at feil slipper til høyere testnivåer

I enkelte tilfeller, spesielt i trinnvise og iterative utviklingsmodeller der kodeendringer pågår nesten kontinuerlig, har automatiserte komponentregresjonstester en sentral rolle i å bygge tillit til at kodeendringer ikke har ødelagt eksisterende funksjonalitet i andre komponenter.

Komponenttesting gjøres ofte isolert fra resten av systemet, avhengig av utviklingsmodellen og andre deler av systemet. Denne tilnærming kan kreve simulatorer, tjenestevirtualisering, testrammeverk, stubber og drivere. Komponenttesting kan både omfatte funksjonelle egenskaper (f.eks korrekthet av beregninger), ikke-funksjonelle egenskaper, f.eks. søk etter minnelekkasjer, og strukturelle egenskaper, f.eks. beslutningstesting.

Testgrunnlag

Eksempler på arbeidsprodukter som kan brukes som testgrunnlag for komponenttesting er:

- Detaljert design
- Kode
- Datamodell
- Komponentspesifikasjoner

Testobjekter

Typiske testobjekter for komponenttesting er:

- Komponenter, enheter eller moduler
- Kode og datastrukturer
- Klasser (kode)
- Databasemoduler

Typiske feil og feilsituasjoner

Eksempler på typiske feil og feilsituasjoner for komponenttesting er:

- Feil funksjonalitet, f.eks. ikke i samsvar med designspesifikasjoner
- Dataflytproblemer
- Feil kode og logikk

Feil blir vanligvis rettet straks de blir avdekket, ofte uten formell feilhåndtering eller administrering. Når utviklere rapporterer feil, gir dette imidlertid viktig informasjon for årsaksanalyse og prosessforbedring.

Spesifikke tilnærminger og ansvar

Komponenttesting blir vanligvis utført av utvikleren som skriver koden, men det krever i det minste tilgang til koden som blir testet. Utviklere kan jobbe både med utvikling og med å finne og rette feil. Utviklere skal som vanlig skrive og utføre tester etter å ha skrevet kode for en komponent. Å designe og skrive tester før den egentlige koden blir skrevet er en av metodene som spesielt kan bli brukt i trinnvise og iterative utviklingsmodeller.

Et eksempel er testdrevet utvikling (Eng: «Test Driven Development (TDD)»). Testdrevet utvikling er svært iterativ og er basert på korte sykluser hvor en først utvikler en automatisert test for en del av funksjonen, og deretter skriver kode som skal levere funksjonaliteten og tester den. Koden blir forbedret til testen lykkes før en ny komponenttest blir utviklet og den eksisterende kodesnutten blir utvidet med ønsket funksjonalitet. Denne syklusen fortsetter til komponenten er fullstendig bygget og alle komponenttestene lykkes. Testdrevet utvikling er et eksempel på en test-først tilnærming. Testdrevet

utvikling har sin opprinnelse i eXtreme Programming (XP) og har spredt seg til andre former for trinnvise og iterative og også sekvensielle livssyklusmodeller (Se: ISTQB-AT Foundation Level Agile Tester Extension Syllabus).

2.2.2 Integrasjonstesting

Mål for integrasjonstesting

Under integrasjonstest fokuserer man på samspill mellom komponenter eller (del)systemer. Målet med integrasjonstesting er:

- Redusere risiko
- Verifisere om funksjonelle og ikke-funksjonelle egenskaper til grensesnittene er realisert som designet og spesifisert
- Bygge tillit til kvalitet av grensesnittene
- Avdekke feil som kan være i grensesnittene selv eller i komponentene eller (del)systemene
- Forebygge at feil blir sluppet til høyere testnivåer

Som med komponenttesting, kan automatiserte integrasjonsregresjonstester i enkelte tilfeller gi tillit til at endringer ikke har ødelagt eksisterende grensesnitt, komponenter eller (del)systemer.

Det er to forskjellige nivåer av integrasjonstesting beskrevet i dette pensumet, som begge kan utføres på testobjekter av varierende størrelse:

- Komponentintegrasjonstest som fokuserer på samspill og grensesnittene mellom komponenter som skal integreres. Komponentintegrasjonstesting utføres etter komponenttesting og er vanligvis automatisert. I trinnvis og iterativ utvikling er komponentintegrasjonstest vanligvis en del av den kontinuerlige integrasjonsprosessen.
- Systemintegrasjonstest som fokuserer på samspill og grensesnittene mellom (del)systemer, applikasjoner og mikrotjenester. Systemintegrasjonstesting kan også omfatte interaksjoner med, og grensesnitt fra, eksterne organisasjoner (f.eks. webtjenester). I dette tilfellet verifiserer den utviklende organisasjonen ikke de eksterne grensesnittene. Dette kan skape ulike utfordringer for testing, f.eks. å sikre at blokkerende feil i den eksterne organisasjonens system er rettet, tilgang til testmiljøer. Systemintegrasjonstesting kan utføres etter systemtesting eller parallelt med systemtestaktiviteter (i både sekvensiell utvikling og trinnvis og iterativ utvikling).

Testgrunnlag

Eksempler på arbeidsprodukter som kan brukes som testgrunnlag for integrasjonstest er:

- Programvare- og systemdesign
- Sekvensdiagrammer
- Spesifikasjoner for grensesnitt og kommunikasjonsprotokoller
- Brukstilfeller
- Arkitektur og design på komponent- eller systemnivå
- Arbeidsflyt eller -prosesser
- Definisjoner av eksterne grensesnitt

Testobjekter

Typiske testobjekter for integrasjonstest er:

- Delsystemer
- Databaser
- Infrastruktur
- Grensesnitt
- API-er
- Mikrotjenester

Typiske feil og feilsituasjoner

Eksempler på typiske feil og feilsituasjoner for komponentintegrasjonstest er:

- Feil data, manglende data eller feil datakoding
- Feil sekvensering eller timing av grensesnittmeldinger
- Konflikt mellom grensesnitt
- Feil i kommunikasjon mellom komponenter
- Manglende feilhåndtering eller kommunikasjonsfeil mellom komponentene
- Feil antagelser om formål, enheter eller grenseverdier til data som sendes mellom komponentene

Eksempler på typiske feil og feilsituasjoner for systemintegrasjonstest er:

- Inkonsistent meldingsstruktur mellom systemene
- Feil data, manglende data eller feil datakoding
- Konflikt mellom grensesnitt
- Feil i kommunikasjon mellom systemer
- Manglende feilhåndtering eller kommunikasjonsfeil mellom systemene
- Feil antagelser om formål, enheter eller grenseverdier til data som sendes mellom systemene
- Manglende samsvar med obligatoriske sikkerhetsforskrifter

Spesifikke tilnærminger og ansvar

Komponent- og systemintegrasjonstester bør konsentrere seg om selve integrasjonen. Hvis man f.eks. integrerer komponent A med komponent B, bør test fokusere på kommunikasjonen mellom disse komponenter og ikke på funksjonaliteten til hver komponent. Funksjonaliteten burde vært verifisert under komponenttesting. Under integrering av system X med system Y, bør test fokusere på kommunikasjonen mellom disse systemer og ikke på funksjonaliteten til hvert system, da denne burde vært verifisert under systemtest. Funksjonelle, ikke-funksjonelle og strukturelle testtyper kan brukes under integrasjonstest.

Utviklere er ofte ansvarlig for komponentintegrasjonstesting. Testere har ofte ansvar for systemintegrasjonstesting. Ideelt sett bør testere som utfører systemintegrasjonstest, kjenne og forstå systemarkitekturen, og de burde ha kunnet påvirke integrasjonsplanleggingen.

Hvis integrasjonstest og integrasjonsstrategi er planlagt før komponenter eller systemer bygges, kan komponentene eller systemene bygges i den rekkefølgen som støtter effektiv testing. Systematiske integrasjonsstrategier kan være basert på systemarkitekturen (top-down eller bottom-up), funksjonelle oppgaver, sekvenser for transaksjonsbehandling eller andre egenskaper til systemet eller komponentene.

For å forenkle isolasjon av feil og avdekke feil så tidlig som mulig, bør integrering normalt være trinnvis (dvs. et lite antall komponenter eller systemer i hver testsyklus eller iterasjon), og ikke som en «big bang» tilnærming (dvs. integrering av alle komponenter eller systemer i ett enkelt trinn). En risikoanalyse av de mest komplekse grensesnittene kan bidra til å rette fokus på de mest kritiske eller viktigste integrasjonene.

Jo større omfanget eller kompleksitet av integrasjonene er, desto vanskeligere blir det å isolere feil til en bestemt komponent eller system. Dette kan føre til økt risiko og ekstra tid for feilsøking. Dette er en grunn til at kontinuerlig integrasjon, hvor programvare blir integrert komponent for komponent (dvs. funksjonell integrasjon), har blitt vanlig praksis. En slik kontinuerlig integrasjon inkluderer ofte automatisert regresjonstesting, ideelt sett på flere testnivåer.

2.2.3 Systemtesting

Mål for systemtesting

Under systemtest fokuserer man på oppførsel og egenskaper til et komplett system eller produkt. Ofte med tanke på ende-til-ende oppgaver som systemet skal kunne støtte og de ikke-funksjonelle oppførsel når disse oppgavene blir utført. Målet med systemtesting er:

- Redusere risiko
- Verifisere om funksjonelle og ikke-funksjonelle oppførsel av systemet er i samsvar med spesifikasjoner og design
- Validere at systemet er komplett og fungerer som forventet
- Bygge tillit til systemets kvalitet som helhet
- Avdekke feil
- Forebygge at feil blir sluppet til senere testnivåer eller til produksjon

For noen systemer kan verifisering av datakvalitet være et mål. Som med komponenttesting og integrasjonstesting, kan automatiserte systemregresjonstester i noen tilfeller gi tillit til at endringene ikke har påvirket eksisterende funksjonalitet eller ende-til-ende muligheter. Systemtest leverer ofte informasjon som brukes av interessenter til å ta beslutninger om leveransene. Systemtest kan også vise samsvar med juridiske eller regulatoriske krav eller standarder.

Testmiljøet bør ideelt sett være mest mulig lik som det endelige produksjonsmiljøet.

Testgrunnlag

Eksempler på arbeidsprodukter som kan brukes som testgrunnlag for systemtest er:

- Kravspesifikasjoner for system og programvare (funksjonell og ikke-funksjonell)
- Risikoanalyserapporter
- Brukstilfeller
- Episke brukerhistorier og brukerhistorier
- Modeller som beskriver systemet
- Tilstandsdiagrammer
- System- og brukerhåndbøker

Testobjekter

Typiske testobjekter for systemtesting er:

- Applikasjoner
- Maskinvare/programvare systemer
- Operativsystemer
- System under test (SUT)
- Systemkonfigurasjon og konfigurasjonsdata

Typiske feil og feilsituasjoner

Eksempler på typiske feil og feilsituasjoner for systemtesting er:

- Beregninger som feiler
- Feil eller uventet funksjonell eller ikke-funksjonell systemoppførsel
- Feil i kontroll- og/eller dataflyt i systemet
- Ikke mulig å teste ende-til-ende funksjonalitet på en korrekt og fullstendig måte
- Systemet fungerer ikke korrekt i driftsmiljø/ene
- Systemet fungerer ikke som beskrevet i system- og brukerhåndbøker

Spesifikke tilnærminger og ansvar

Systemtest bør fokusere på en fullstendig, ende-til-ende oppførsel av systemet som helhet, både funksjonelt og ikke-funksjonelt. Systemtest bør bruke de mest hensiktsmessige teknikkene (se kapittel 4) for aspektene av systemet som skal testes. F.eks. kan en beslutningstabell opprettes for verifisering om funksjonell oppførsel er beskrevet som i forretningsreglene.

Systemtest blir vanligvis utført av uavhengige testere. Feil i spesifikasjoner, f.eks. manglende brukerhistorier, feil oppgitte forretningskrav, etc. kan føre til manglende forståelse for eller uenighet om forventet systemoppførsel. Slike situasjoner kan forårsake falske positive og falske negative observasjoner, som koster tid og reduserer effektiviteten til feilavdekkingen. Tidlig deltakelse av testere i forbedring av brukerhistorier eller andre statiske testaktiviteter, f.eks. reviews, bidrar til begrensning av slike situasjoner (se kapittel 3).

2.2.4 Akseptansetest

Mål med akseptansetest

Akseptansetest fokuserer, slik som systemtest, vanligvis på oppførsel og egenskaper til et helt system eller produkt. Mål med akseptansetest er:

- Få tillit til kvaliteten på systemet som helhet
- Validere at systemet er komplett og vil fungere som forventet
- Verifisere at funksjonelle og ikke-funksjonelle egenskaper er implementert som spesifisert

Akseptansetest skal gi informasjon for å kunne vurdere om systemet er klart for leveranse og klart for bruk av kunden (sluttbruker). Man skal kunne avdekke feil under akseptansetest, men å finne feil er ofte ikke et mål. Hvis man avdekker et betydelig antall feil under akseptansetest, kan det i noen tilfeller betraktes som en stor prosjektrisiko. Akseptansetest kan også oppfylle krav som følge av lovverk eller pålagte standarder.

Vanlige former for akseptansetest kan være:

- Brukerakseptansetest
- Driftsakseptansetest
- Akseptansetesting av samsvar med kontrakt, lover og regler
- Alfa- og betatest

Disse former for akseptansetest blir i de følgende avsnittene beskrevet.

Brukerakseptansetest (UAT)

Akseptansetest av systemet utført av brukere er vanligvis fokusert på validering av brukbarheten av systemet i et reelt eller simulert driftsmiljø. Det er viktig at de som skal bruke systemet deltar i testen. Hensikten er å bygge tillit til at systemet møter brukernes behov, at det oppfyller kravene, og kan støtte forretningsprosessene med et minimum av problemer, kostnader og risiko.

Driftsakseptansetest (OAT)

Akseptansetest av systemet utført av personell for drift og/eller systemadministrasjon tester vanligvis i et (simulert) produksjonsmiljø. Tester fokuserer på operasjonelle aspekter, og kan omfatte:

- Test av sikkerhetskopiering og gjenoppretting
- Installering, avinstallering og oppgradering
- Gjenopprettelse etter katastrofe
- Brukeradministrasjon
- Vedlikeholdsoppgaver
- Innlasting av data og migreringsoppgaver
- Sikkerhetstest
- Ytelsestest

Hovedmålet med driftsakseptansetesting er å få tillit til at systemadministratorene kan sørge for at systemet fungerer korrekt for sluttbrukerne i driftsmiljøet, selv under eksepsjonelle eller vanskelige forhold.

Akseptansetesting av samsvar med kontrakt, lover og regler

Akseptansetesting av samsvar med kontraktenutføres mot akseptansekriterier som er spesifisert i kontrakten for produksjon av spesialutviklet programvare. Disse akseptansekriteriene bør defineres når partene blir enig om kontrakten. Akseptansetesting av samsvar med kontrakten utføres ofte av brukere eller av uavhengige testere.

Akseptansetest av samsvar med lover og regler utføres mot eventuelle forskrifter som er gjeldende for systemet, f.eks. offentlige, juridiske eller sikkerhetsrelevante lover og retningslinjer. Testen utføres ofte av sluttbrukere eller av uavhengige testere, noen ganger overvåket eller revidert av de aktuelle myndigheter.

Hovedformålet med denne testen er å bygge tillit til at kontrakts- og forskriftskrav er oppfylt.

Alfa- og betatest

Alfa- og betatesting brukes vanligvis av utviklere av hyllevare for å få tilbakemelding fra potensielle eller eksisterende brukere, kunder og/eller operatører før løsningen settes ut på markedet. Alfatesting blir utført i leverandørens testmiljø, ikke av utviklingsteamet, men av mulige eller eksisterende kunder og/eller operatører eller et uavhengig testteam. Betatesting utføres av potensielle eller eksisterende kunder

og/eller operatører i sine egne miljø. Betatesting kan gjennomføres etter alfatesting, eller kan bli gjennomført uten at noen foregående alfatesting har skjedd.

Et mål med alfa- og betatesting er å bygge tillit hos potensielle eller eksisterende kunder og/eller operatører til at de kan bruke systemet under normale, hverdagslige forhold og i et realistisk produksjonsmiljø for å oppnå kundenes mål med minimale problemer, kostnader og risiko. Et annet mål kan være å avdekke feil som er relatert til typiske forhold og miljø(er) der systemet skal brukes, spesielt når disse forholdene og miljøene er vanskelige å simulere av utviklingsteamet.

Testgrunnlag

Eksempler på arbeidsprodukter som kan brukes som testgrunnlag for alle typer akseptansetest er:

- Forretningsprosesser
- Bruker- eller forretningskrav
- Forskrifter, juridiske kontrakter og standarder
- Brukstilfeller
- Brukerhistorier
- Systemrelaterte krav
- System- eller brukerdokumentasjon
- Installasjonsprosedyrer
- Risikoanalyserapporter

I tillegg kan et eller flere av følgende arbeidsprodukter brukes som testgrunnlag for å avlede testtilfeller for driftsakseptansetesting:

- Rutiner for sikkerhetskopiering og gjenoppretting
- Rutiner for gjenopprettelse etter katastrofe
- Ikke-funksjonelle krav
- Driftsdokumentasjon
- Distribuering- og installasjonsinstruksjoner
- Ytelsesrelaterte krav
- Databasepakker
- Sikkerhetsstandarder eller forskrifter

Typiske testobjekter

Typiske testobjekter for enhver form for akseptansetest er:

- System under test (SUT)
- Systemkonfigurasjon og konfigurasjonsdata
- Forretningsprosesser for et fullt integrert system
- Metoder for gjenoppretting og «hot standby» miljøer (for kontinuitet i drift og testing av gjenoppretting etter katastrofe)
- Operasjonelle og vedlikeholds- eller forvaltningsprosesser

- Skjemaer
- Rapporter
- Eksisterende og konverterte produksjonsdata

Typiske feil og feilsituasjoner

Eksempler på typiske feil for enhver form for akseptansetest er:

- Systemets arbeidsprosesser er ikke i samsvar med forretnings- eller brukerkrav
- Forretningsregler er ikke implementert på riktig måte
- Systemet er ikke i samsvar med kontraktsmessige- eller lovpålagte krav
- Ikke-funksjonelle feilsituasjoner relatert til sikkerhet eller ytelse under høy belastning, eller feil oppførsel på en støttet plattform

Spesifikke tilnærminger og ansvar

Akseptansetest blir ofte gjennomført av kunder, forretningsbrukere, produkteiere eller operatører av systemet, men andre interessenter kan også være involvert.

Akseptansetest regnes ofte som det siste testnivået i en sekvensiell utviklingslivssyklus, men det kan også blir gjennomført i andre sammenheng, f.eks.:

- Akseptansetest av et hyllevareprodukt kan bli gjennomført etter installering eller integrering
- Akseptansetest av en mindre funksjonell forbedring kan bli gjennomført uten systemtest

I trinnvis og iterativ utvikling kan teamet benytte ulike former for akseptansetesting under og ved slutten av hver iterasjon. F.eks. tester som fokuserer på verifisering av ny funksjon mot akseptansekriterier og tester som fokuserer på validering av at en ny funksjon oppfyller brukernes behov. I tillegg kan alfatester og betatester bli gjennomført, enten ved slutten av hver iterasjon, etter ferdigstillelse av hver iterasjon, eller etter et antall iterasjoner. Brukerakseptansetest, driftsakseptansetesting, akseptansetest av samsvar med kontrakt og akseptansetest av samsvar med lover og regler kan også bli gjennomført enten ved slutten av hver iterasjon, etter fullføring av hver iterasjon, eller etter et antall iterasjoner.

2.3 Testtyper

En testtype er en gruppe testaktiviteter med fokus på verifisering av spesifikke egenskaper for systemet, eller en del av systemet, basert på spesifikke testmål. Slike mål kan være:

- Evaluering av funksjonelle kvalitetsegenskaper, slik som fullstendighet, korrekthet og hensiktsmessighet
- Evaluering av ikke-funksjonelle kvalitetsegenskaper, slik som pålitelighet, ytelse, sikkerhet, kompatibilitet og brukervennlighet
- Vurdering om detaljert design eller arkitektur til komponentene eller systemet er korrekt, komplett og som spesifisert
- Evaluering av effektene av endringer, f.eks. bekreftelse på at feilene er blitt rettet (retesting) og leting etter utilsiktede endringer i oppførsel som følge av programvare- eller miljøendringer (regresjonstesting)

2.3.1 Funksjonell testing

Funksjonell testing av et system omfatter tester som verifiserer funksjoner som systemet skal utføre. Funksjonelle krav kan beskrives i arbeidsprodukter som f.eks. spesifikasjoner av forretningskrav, episke brukerhistorier, brukerhistorier, brukstilfeller eller andre typer funksjonelle spesifikasjoner. Det kan også finnes udokumentert krav. Mer generelt: funksjoner er "hva" systemet skal gjøre.

Funksjonelle tester bør utføres på alle testnivåer, f.eks. kan komponenttester være basert på en komponentspesifikasjon, selv om fokus er forskjellig på hvert nivå (se avsnitt 2.2).

Funksjonell testing evaluerer hvordan programvaren oppfører seg. Svartboks teknikker kan bli brukt for å avlede funksjonelle testbetingelser og testtilfeller for komponenter eller systemet (se avsnitt 4.2).

Grundigheten av funksjonell testing kan måles gjennom funksjonell testdekning. Funksjonell testdekning er spesifisert som i hvilken grad en type funksjonelt element har blitt inkludert i tester, og uttrykkes som en prosentandel av typen/elementene som dekkes. F.eks. ved å ha sporbarhet mellom testtilfeller og funksjonelle krav, kan prosentandelen av kravene som er dekket av testene beregnes, og en kan potensielt identifisere områder som ikke blir testet nok.

Funksjonell testdesign og testutføring kan kreve spesielle ferdigheter eller kunnskaper. Domenekunnskap kan kreves om det spesielle forretningsproblemet som løses av programvaren (f.eks geologisk modelleringsprogramvare for olje- og gassindustrien) eller den spesielle rollen som programvaren har, f.eks. interaktiv underholdning.

2.3.2 Ikke-funksjonell testing

Ikke-funksjonell testing av et system evaluerer kvalitetsegenskaper til systemer og programvare som brukbarhet, ytelse eller sikkerhet. Se ISO/IEC 25010 standarden for en klassifisering av programvarekvalitetsegenskaper. Ikke-funksjonell testing er testing av «hvor godt» funksjonalitet blir levert til sluttbrukere.

I motsetning til vanlige misoppfatninger, kan (og ofte bør) ikke-funksjonell testing utføres på alle testnivåer, og skal helst gjøres så tidlig som mulig. Sen oppdagelse av ikke-funksjonelle feil kan være ekstremt ødeleggende for prosjektets suksess.

Svartboks teknikker (se avsnitt 4.2) kan brukes til generering av testbetingelser og testtilfeller for ikke-funksjonell testing. F.eks. kan grenseverdianalyse brukes til definering av stressbetingelsene for ytelsestest.

Grundigheten til ikke-funksjonell testing kan måles gjennom ikke-funksjonell testdekning. Ikke-funksjonell testdekning defineres som i hvilken grad en type ikke-funksjonelle egenskaper har blitt inkludert i tester, og uttrykkes som en prosentandel av egenskapene som ble testet. F.eks. ved å bruke sporbarhet mellom testtilfeller og støttede enheter for en mobilapplikasjon, kan prosentandelen enheter som er testet for kompatibilitet beregnes, og eventuell manglende testdekning kan bli identifisert.

Ikke-funksjonell testdesign og -utføring kan kreve spesielle ferdigheter eller kunnskaper, f.eks. kunnskap om de implisitte svakhetene i design eller teknologi f.eks. sikkerhetsproblemer knyttet til bestemte programmeringsspråk eller spesielle egenskaper til f.eks. brukere av helsevesenets styringssystemer.

Se ISTQB-ATA Advanced Level Test Analyst Syllabus, ISTQB-ATTA Advanced Level Technical Test Analyst Syllabus, ISTQB-SEC Advanced Level Security Tester Syllabus, og andre ISTQB spesialistmoduler for mer informasjon om test av ikke-funksjonelle kvalitetsegenskaper.

2.3.3 Hvitboks testing

Med hvitboks testing avleder man testtilfeller basert på systemets eller programvarens interne struktur eller implementering. Kode, arkitektur, arbeidsflyt og/eller dataflyt i systemet kan brukes for analyse av strukturene (se avsnitt 4.3).

Grundigheten av hvitboks testing kan måles med beregning av strukturell testdekning. Strukturell testdekning er definert som i hvilken grad en komponent eller enhet har blitt inkludert i tester, og uttrykkes som en prosentandel tilsvarende av det som ble testet.

I komponenttest er kodedekningen basert på prosentandelen av komponentkoden som har blitt testet. Det kan måles når det gjelder forskjellige elementer av koden, f.eks. prosentandelen av instruksjoner eller beslutninger som ble testet. Disse typer testdekning faller generelt under termen kodedekning.

I komponentintegrasjonstest kan hvitboks testing være basert på systemets arkitektur, som f.eks. grensesnittene mellom komponenter, og testdekning kan måles som prosentandelen av totalt antall grensesnitt som er dekket av testene.

Hvitboks testdesign og -utføring kan kreve spesielle ferdigheter eller kunnskaper. F.eks. hvordan koden er bygget (f.eks. for å bruke kodedekningsverktøy), hvordan data blir lagret (f.eks. for å evaluere mulige databaseoppslag) og hvordan en korrekt tolker resultatene.

2.3.4 Endringsrelatert testing

Når det gjøres endringer i et system, enten for å rette en feil eller på grunn av ny eller endret funksjonalitet, bør man verifisere at endringene har rettet feilen eller implementert ny funksjonalitet på riktig måte, og ikke har hatt noen uforventede negative konsekvenser.

- Retesting (eller bekreftelsestesting): Etter at en feil er rettet, kan den nye versjonen av programvaren bli testet igjen med alle testtilfeller som opprinnelig feilet. Programvaren kan også bli testet med nye tester, hvis koden som feilet før hadde funksjonelle mangler. Som minimum må testtrinnene for å reprodusere feilen utføres på den nye programvareversjonen. Hensikten med en retest er å bekrefte at den opprinnelige feilen har blitt rettet.
- Regresjonstesting: Det er mulig at en endring som er utført i en del av koden, enten en utbedring eller en annen type endring, ved uhell resulterer i negativ oppførsel i andre deler av koden. Det kan hende enten i samme komponent, i andre komponenter i samme systemet, eller til og med i andre systemer. Endringene kan også omfatte endringer i miljøet, f.eks. en ny versjon av et operativsystem eller databasesystem. Slike utilsiktede bivirkninger kalles regresjoner, og regresjonstest skal finne disse.

Retesting og regresjonstesting utføres på alle testnivåer.

Spesielt i trinnvise og iterative livssykluser vil nye funksjoner, endringer i eksisterende funksjoner og restrukturering av kode resultere i hyppige kodeendringer, som også krever endringsrelatert testing. På grunn av systemets utvikling er retesting og regresjonstesting svært viktig. Dette er spesielt relevant for Tingens Internett der enkelte objekter eller enheter ofte oppdateres eller erstattes.

Regresjonstestsuiter kjøres mange ganger og er ganske stabile. Derfor er regresjonstest en sterk kandidat for automatisering. Automatisering av disse testene bør starte tidlig i prosjektet (se kapittel 6 for verktøystøtte).

2.3.5 Forholdet mellom testtyper og testnivåer

Alle testtyper beskrevet ovenfor kan brukes på alle testnivåer. Dette illustreres med test av en bankapplikasjon under.

Følgende er eksempler på funksjonelle tester:

- I komponenttest testes hvordan en komponent skal beregne rente
- I komponentintegrasjonstest testes hvordan kontoinformasjon, som er lagt inn av bruker, sendes til forretningslogikken
- I systemtest testes hvordan kontoinnehavere kan søke om kreditt
- I systemintegrasjonstest testes hvordan systemet bruker en ekstern mikrotjeneste for å sjekke kontoinnehavers kredittverdighet
- I akseptansetest testes hvordan banken godkjenner eller avviser kredittkortsøknaden

Følgende er eksempler på ikke-funksjonelle tester:

- I komponenttest er ytelsestestene utformet for å evaluere antall CPU-sykluser som kreves for å utføre en kompleks totalrenteberegning
- I komponentintegrasjonstest er sikkerhetstestene designet for å avdekke sikkerhetsproblemer med «buffer overflow» på grunn av data som sendes fra brukergrensesnittet til forretningslogikken
- I systemtest er portabilitetstestene utformet for å kontrollere om brukergrensesnitt fungerer på alle spesifiserte nettlesere og mobile enheter
- I systemintegrasjonstest er pålitelighetstestene utformet for å evaluere systemets robusthet hvis mikrotjeneste for kredittvurdering ikke svarer
- I akseptansetest er brukervennlighetstestene designet for evaluering av tilgjengeligheten for funksjonshemmede

Følgende er eksempler på hvitboks tester:

- I komponenttest testes for å oppnå fullstendig programinstruksjons- og beslutningsdekning (se avsnitt 4.3) for alle komponenter som utfører økonomiske beregninger
- I komponentintegrasjonstest testes for å verifisere hvordan hver skjerm i nettlesergrensesnittet overfører data til neste skjerm og til forretningslogikken
- I systemtest testes sekvenser av nettsider som kan bli vist under en søknadsprosess for kredittkort
- I systemintegrasjonstest testes mulige forespørselstyper sendt til mikrotjenesten for kredittvurdering
- I akseptansetest testes støttede økonomiske datafilstrukturer og verdier for bankoverføringer

Følgende er eksempler for endringsrelaterte tester:

- I komponenttest er det bygd automatiserte regresjonstester for hver komponent, og disse inngår i det kontinuerlige integrasjonsrammeverket
- I komponentintegrasjonstest er testene designet for å bekrefte utbedringer til brukergrensesnittet når disse kodeendringene blir integrert med eksisterende kode

- I systemtest utføres alle testene igjen for en gitt arbeidsprosess når grensesnittdetaljer på den aktuelle arbeidsflyten endres
- I systemintegrasjonstest gjennomføres tester av applikasjonen som inkluderer mikrotjeneste for kredittvurdering som en del av den kontinuerlige distribusjonen av denne mikrotjenesten
- I akseptansetest blir alle tidligere mislykkede testtilfeller retestet etter at en feil funnet i akseptansetest er rettet

Mens dette avsnittet gir eksempler på hver testtype på tvers av nivåene, er det ikke nødvendig for alle utviklingsprosjekter å bruke alle testtyper representert på alle nivåer. Det er imidlertid viktig å kjøre de spesifiserte testtyper på hvert nivå, spesielt på det tidligste nivået for denne testtypen.

2.4 Vedlikeholdstesting

Når programvaren er installert i produksjonsmiljøet, må programvare og systemer vedlikeholdes. Endringer er nesten uunngåelige etter at programvare og systemer er levert. Det kan være å rette mangler oppdaget i operativ bruk, legge til eller endre/slette eksisterende funksjonalitet. Vedlikehold er også nødvendig for å bevare eller forbedre ikke-funksjonelle kvalitetsegenskaper for komponenten eller systemet i hele livssyklusen, spesielt ytelse, pålitelighet, sikkerhet, kompatibilitet og portabilitet.

Når det gjøres endringer bør man planlegge vedlikeholdstesting. Dette for å evaluere endringene og for å kontrollere mulige bivirkninger (regresjoner) i de uendrede delene av systemet (hvilket vanligvis er det meste av systemet). Vedlikeholdstest fokuserer på test av endringene i systemet, samt test av uendrede deler som kan ha blitt påvirket av endringene. Vedlikehold kan innebære planlagte leveranser og ad-hoc endringer («hot fixes», nødendringer).

En vedlikeholdsleveranse kan kreve vedlikeholdstesting på flere testnivåer og inkludere ulike testtyper avhengig av omfanget. Se spesielt avsnitt 2.3.4 om endringsrelatert testing.

Omfanget av vedlikeholdstesting er avhengig av:

- Graden av risiko for endringen, f.eks. i hvilken grad det endrede området av programvaren kommuniserer med andre komponenter eller systemer
- Størrelsen på det eksisterende systemet
- Størrelsen på endringen

2.4.1 Årsaker til vedlikehold

Det er flere årsaker til vedlikehold og dermed vedlikeholdstesting, både for planlagte og ikke-planlagte endringer.

Vi kan klassifisere vedlikeholdsårsakene som følger:

- Endringsbehov: Planlagte forbedringer, korrigerende- og hasteendringer, endringer i driftsmiljøet (dvs. planlagte oppgraderinger av operativsystem eller database), oppgraderinger av hyllevare og oppdateringer for å rette feil og fjerne sårbarheter
- Migrering: Fra en plattform til en annen, noe som kan kreve operasjonelle tester i det nye miljøet og av den endrede programvaren. Ellers kan det kreves tester av datakonvertering når data fra en annen applikasjon blir migrert inn i systemet
- Utfasing (avinstallering): Når et program eller system blir faset ut, kan dette kreve testing av datamigrering eller arkivering dersom det er behov for datalagring. Det kan også være nødvendig

å teste back-up og gjenopprettingsprosesser for å sikre at data fortsatt er tilgjengelig. I tillegg kan det være behov for regresjonstesting for å sikre at funksjonalitet som forblir i drift fortsatt fungerer

For Tingenes Internett kan vedlikeholdstesting bli nødvendig ved innføring av helt nye eller modifiserte elementer, f.eks. maskinvareenheter og programvaretjenester, inn i det totale systemet. Vedlikeholdstest for slike systemer legger særlig vekt på integrasjonstesting på forskjellige nivåer (f. eks. nettverksnivå og applikasjonsnivå) og på sikkerhetsaspekter, særlig når personopplysninger er berørt.

2.4.2 Konsekvensanalyse ved vedlikehold

Konsekvensanalyse bør gjennomføres i sammenheng med en vedlikeholdsleveranse. Dette for å identifisere planlagte konsekvenser, og forventede og mulige bivirkninger av en endring, og for å identifisere områdene i systemet som vil bli påvirket av endringen. Konsekvensanalyse kan også bidra til å identifisere hvordan en endring påvirker eksisterende tester. Effektene og de berørte områdene i systemet bør regresjonstestes, muligens etter oppdatering av eksisterende test som påvirkes av endringen.

Konsekvensanalysen skal helst utføres før en endring blir implementert, for å kunne beslutte om endringen i det hele tatt skal gjennomføres.

Konsekvensanalyse kan være vanskelig hvis:

- Spesifikasjoner er utdatert eller mangler, f.eks. forretningskrav, brukerhistorier, arkitektur
- Testtilfeller ikke er dokumentert eller er utdatert
- To-veis sporbarhet mellom test og testbasis har mangler
- Verktøystøtte er vanskelig eller ikke mulig
- De involverte personene har ikke (nok) domene- og/eller systemkunnskap
- Under utvikling har man ikke tenkt på vedlikeholdbarhet

3 Statisk test — 135 minutter

Nøkkelord

ad hoc gransking, dynamisk testing, formell review, inspeksjon, perspektivbasert lesing, review, rollebasert review, scenariobasert review, sjekklistebasert review, statisk analyse, statisk testing, teknisk review, uformell review, walkthrough (gjennomgang)

OBS: Ordene review, gjennomgang og gransking brukes om hverandre og i samme betydning. Det engelske ordet walkthrough brukes også på norsk.

Læremål

3.1 Grunnleggende fakta om statisk testing

FL-3.1.1 (K1) Kjenn til typer av arbeidsprodukter der de ulike statiske testteknikker kan brukes til undersøkelse

FL-3.1.2 (K2) Bruk eksempler for å beskrive verdien av statisk testing

FL-3.1.3 (K2) Forklar forskjellen mellom statiske og dynamiske teknikker, ved å sammenligne mål, typer feil som skal finnes og rollen disse teknikkene har i programvarens livssyklus

3.2 Reviewprosessen

FL-3.2.1 (K2) Oppsummer aktivitetene i reviewprosessen

FL-3.2.2 (K1) Kjenn de ulike rollene og ansvarsområdene i en formell review

FL-3.2.3 (K2) Forklar forskjellene mellom følgende reviewtyper: uformell review, walkthrough, teknisk review og inspeksjon

FL-3.2.4 (K3) Bruk en reviewteknikk på et arbeidsprodukt for å finne feil

FL-3.2.5 (K2) Forklar faktorene som bidrar til en suksessfull review

3.1 Grunnleggende fakta om statisk testing

I motsetning til dynamisk testing, som krever at programvaren utføres, er statisk testing manuell undersøkelse av arbeidsprodukter (review) eller verktøydrevet evaluering av kodeneller andre arbeidsprodukter (statisk analyse). Begge typer statisk testing evaluerer testobjektet uten å utføre det.

Statisk analyse er viktig for sikkerhetskritiske systemer, f.eks. programvare i fly, medisin, eller kjernekraft, men statisk analyse har også blitt viktig og vanlig i andre sammenhenger. F.eks. er statisk analyse en viktig del av sikkerhetstesting. Statisk analyse blir også ofte inkludert i automatiske systemer for bygging og levering, f.eks. i smidig utvikling, kontinuerlig levering og kontinuerlig utvikling.

3.1.1 Arbeidsprodukter der statisk testing kan brukes

Nesten hvert arbeidsprodukt kan undersøkes ved hjelp av statisk testing (reviews og/eller statisk analyse), f.eks.:

- Spesifikasjoner, inklusive forretningskrav, funksjonelle krav, sikkerhetskrav
- Episke brukerhistorier, brukerhistorier og akseptansekriterier
- Arkitektur- og designspesifikasjon
- Kode
- Testmateriell, som kan omfatte bl.a. testplaner, testtilfelle, testprosedyrer og automatiserte testskript
- Brukermanualer
- Websider
- Kontrakter, prosjektplaner, tidsplaner og budsjetter
- Modeller, som f.eks. aktivitetsdiagrammer, som kan bli brukt for modellbasert testing (se ISTQB-MBT Foundation Level Model-Based Tester Extension Syllabus og Kramer 2016)

Reviews kan brukes for ethvert arbeidsprodukt som deltakerne kan lese og forstå. Statisk analyse kan effektivt brukes til ethvert arbeidsprodukt med en formell struktur (typisk kode eller modeller) der et passende verktøy for statisk analyse eksisterer. Statisk analyse kan til og med brukes med verktøy som evaluerer arbeidsprodukter skrevet i naturlig språk som krav, f.eks. for å sjekke stavelse, grammatikk og lesbarhet.

3.1.2 Nytten av statisk testing

Statiske testteknikker er nyttige på mange måter. Når en bruker dem tidlig i livssyklusen, kan de hjelpe med å finne feil før dynamisk testing utføres. Dette skjer f.eks. ved review av krav, design eller i gjennomgang av produktkøen. Feil som finnes tidlig er ofte mye billigere å rette enn feil funnet senere (grunnprinsipp fra kapittel 1), spesielt når en sammenligner med feil funnet under bruk av programvaren. Å bruke statiske testteknikker for å finne feil og så å rette dem med en gang er nesten alltid mye billigere enn å bruke dynamisk testing for å finne feil i testobjektet og så å rette dem. Spesielt når man tenker på alle ekstrakostnaden for å oppdatere andre arbeidsprodukter og for å utføre retesting og regresjonstesting.

Flere nytteeffekter av statisk testing kan omfatte:

- Finne og rette feil er mer effektivt og skjer før en utfører dynamisk test
- Finne feil som er vanskelig å finne med dynamisk testing

- Forebygge feil i design eller kode ved å oppdage inkonsistens, motsigelser, uklarheter, forglemmelser, unøyaktigheter og redundans i krav
- Øke produktiviteten i utviklingen, f.eks. pga. forbedret design eller kode som er lettere å vedlikeholde
- Redusere kostnad og tid for utviklingen og testingen
- Redusere total kvalitetskostnad for hele programvarens livssyklus pga. mindre feil senere under utvikling eller etter levering
- Forbedre kommunikasjonen mellom medlemmene i gruppen ved at de deltar i reviews

3.1.3 Forskjeller mellom statisk og dynamisk testing

Statisk testing og dynamisk testing kan tjene de samme formål (se avsnitt 1.1.1), som å muliggjøre bedømmelse av kvaliteten til arbeidsproduktene og å finne feil så tidlig som mulig. Statisk og dynamisk testing utfyller hverandre ved å finne ulike typer feil.

En hovedforskjell er at statisk testing finner feil i arbeidsprodukter direkte istedenfor å finne «failures» som er forårsaket av «defects» når programvaren utføres. En «defect» kan være i et arbeidsprodukt i lang tid uten å forårsake en synlig feil. Forgreningen i programmet der feilen ligger blir kanskje sjelden utført eller er vanskelig å nå. Dermed blir det ikke enkelt å lage og utføre en dynamisk test som finner feilen. Statisk testing kan finne en slik feil med mye mindre arbeid.

En annen forskjell er at statisk testing kan brukes til å forbedre konsistensen og den interne kvaliteten av et arbeidsprodukt, mens dynamisk testing typisk fokuserer på eksternt synlig oppførsel.

Sammenlignet med dynamisk testing er bl.a. følgende feil typisk enklere og billigere å finne og rette ved hjelp av statisk testing:

- Feil i krav, f.eks. inkonsistenser, uklarheter, motsigelser, utelatelser, unøyaktigheter og unødig gjentakelse
- Feil i design, f.eks. ineffektive algoritmer eller databasestrukturer, høy kobling, lavt samhold
- Feil i kode, f.eks. variabler med udefinerte verdier, variabler som blir deklarert men aldri brukt, kode som ikke kan utføres, duplisert kode
- Avvik fra standarder, f.eks. kodestandard
- Feil i spesifikasjoner av grensesnitt, f.eks. ulike måleenheter på de to sidene av et grensesnitt
- Sårbarheter vedr. sikkerheten, f.eks. sårbarhet pga. «buffer overflow»
- Hull eller unøyaktigheter i sporbarhet eller testdekning av testbasisen, f.eks. tester som mangler for et akseptansekriterium

I tillegg kan de fleste problemer med vedlikeholdbarhet bare finnes ved hjelp av statisk testing (f.eks. dårlig modulinndeling, dårlig gjenbrukbarhet for komponenter, kode som er vanskelig å analysere og endre uten å introdusere nye feil).

3.2 Reviewprosessen

Reviews varierer fra uformell til formell. Uformelle reviews følger ingen definert prosess og har ikke formelt dokumentert resultat. Dvs. de blir bare utført. Formelle reviews er karakterisert ved at en gruppe deltar, at resultatene er dokumentert og at det følges dokumenterte prosedyrer for utførelsen av selve review. Reviewprosessens formalitet avhenger av faktorer som utviklingsmodellen, utviklingsprosessens

modenhet, arbeidsproduktets kompleksitet, juridiske krav, myndighetskrav og/eller behovet for at gjennomførelsen av selve review kan kontrolleres (revisjonsspor).

Fokus for en review er avhengig av målsetningene en har blitt enig om. Eksempler er å finne feil, å forstå materialet, å lære opp deltakerne eller å diskutere og beslutte ved konsensus. Eksempler på deltakere som skal læres opp er testere og nye gruppemedlemmer.

ISO standard (ISO/IEC 20246) har mer dyptgående beskrivelser av reviewprosessen samt roller og reviewteknikker.

3.2.1 Reviewprosessen

Reviewprosessen omfatter følgende hovedaktiviteter:

Planlegging

- Definere omfanget, som omfatter formål med denne review, hvilke dokumenter eller deler av dokumenter som skal gjennomgås, og kvalitetsegenskaper som skal evalueres
- Estimere arbeidsinnsats og tid
- Identifisere egenskaper som f.eks. type review med roller, aktiviteter og sjekklister
- Velge ut hvem som skal delta og tildele roller
- Definere start- og sluttkriterier for mer formelle reviewtyper (som inspeksjoner)
- Kontrollere at startkriteriene er oppfylt (for mer formelle reviewtyper)

Oppstart

- Dele ut arbeidsproduktet (fysisk eller elektronisk) og annet materiale, som formularer for å melde feil, sjekklister og relaterte arbeidsprodukter
- Forklare omfang, prosess, roller og arbeidsproduktene til deltakerne
- Svare på spørsmål som deltakerne måtte ha om selve review

Individuell review (individuell forberedelse)[1]

- Gjennomgå hele arbeidsproduktet eller deler av det
- Notere mulige feil, anbefalinger og spørsmål

Kommunikasjon og analyse av observasjoner eller feil

- Kommunisere mulige feil, f.eks. i et reviewmøte
- Analysere mulige feil ved å tildele ansvar for å rette dem samt gi dem en status
- Evaluere og dokumentere kvalitetsegenskapene til arbeidsproduktet
- Evaluere reviewresultater mot sluttkriteriene for å bestemme resultatet (underkjent; vesentlige endringer påkrevd; godkjent; mindre endringer påkrevd)

[1] Dette er den viktigste delen av en review! En må sammenligne arbeidsproduktet med spesifikasjon og krav og eventuelle forskrifter og standarder. En må bruke eventuelle sjekklister eller sjekke det en vet må sjekkes. En må bruke nok tid til å gjøre dette grundig. Leseteknikker er bl.a. beskrevet i avsnitt 3.2.4.

Retting og rapportering

- Lage feilrapporter for alt som krever endringer
- Rette feil som er funnet (gjøres typisk av forfatteren) i det gjennomgåtte arbeidsproduktet
- Kommunisere feil til den rette personen eller det rette teamet (for feil som er funnet i relaterte arbeidsprodukter)
- Oppdatere feilstatus (i formelle reviews) og eventuelt sjekke at den som påpekte feilen er enig
- Samle måledata (for mer formelle reviewtyper)
- Kontrollere at sluttkriterier er oppfylt (for mer formelle reviewtyper)
- Godkjenne arbeidsproduktet når sluttkriteriene er oppfylt

Resultatene for en review varierer avhengig av reviewtypen og graden av formalitet slik som beskrevet i avsnitt 3.2.3.

3.2.2 Roller og ansvarsområder i en formell review

En typisk formell review omfatter rollene nedenfor:

Forfatter

- Lager arbeidsproduktet som skal gjennomgås
- Retter feil i arbeidsproduktet etter review (hvis nødvendig)

Ledelse

- Er ansvarlig for planlegging av review
- Beslutter om review skal gjennomføres
- Tildeler personer, tid og budsjett
- Holder øye med kostnadseffektiviteten
- Tar affære i tilfelle dårlige resultater

Møteleder (ofte kalt moderator)

- Sikrer effektive reviewmøter (hvis de holdes)
- Megler, hvis nødvendig, mellom ulike synspunkter
- Moderatoren er ofte avgjørende for suksessen av en review

Reviewleder

- Har det overordnete ansvaret for en review
- Beslutter hvem som skal delta og organiserer når og hvor det vil finne sted

Reviewere / revisorer / inspektører

- Kan være eksperter på fagområdet, personer som arbeider i prosjektet, personer som er interessent i arbeidsproduktet, og/eller personer med spesifikk teknisk eller forretningsmessig bakgrunn
- Identifiserer mulige feil i arbeidsproduktet under review

- Kan representere ulike perspektiver, f.eks. tester, programmerer, bruker, operatør, forretningsspesialist, ekspert for brukervennlighet osv.

Sekretær (den som skriver protokollen)

- Samler og kombinerer rapporterte feil som er funnet under individuell review
- Dokumenterer nye mulige feil, anmerkninger, problemer, åpne punkter og beslutninger fra møtet (hvis det blir holdt)

I noen reviewtyper kan en person fylle mer enn en rolle, og handlingene som hører til hver rolle kan også variere avhengig av reviewtypen. I tillegg har verktøy for reviewstøtte blitt tilgjengelig, spesielt for å loggføre feil, åpne punkter og beslutninger. Der en bruker verktøy forsvinner ofte behovet for en protokollfører (sekretær).

Flere andre og mer detaljerte roller er mulige, se ISO standard (ISO/IEC 20246).

3.2.3 Reviewtyper

Selv om reviews kan brukes til ulike formål er et av hovedmålene å finne feil. Alle reviewtyper kan hjelpe til å finne feil. Hvilken reviewtype en velger bør en basere bl.a. på prosjektets behov, tilgjengelige ressurser, produkttype og risiko, anvendelsesområde og bedriftskultur.

Reviews kan klassifiseres på ulike måter. Det følgende lister opp de fire mest vanlige reviewtypene og deres egenskaper.

Uformell review, f.eks. fagfellevurdering, parvis arbeid

- Hovedmål: Finne mulige feil
- Mulige tilleggsmål: Generere nye ideer eller løsninger, løse mindre problemer fort
- Ikke basert på en formell (dokumentert) prosess
- Kan utelate reviewmøtet
- En kollega av forfatteren (fagfellevurdering) eller flere personer kan sjekke
- Resultater kan bli dokumentert (men må ikke)
- Nytteeffekten varierer avhengig av reviewere
- Valgfri bruk av sjekklister
- Veldig vanlig i smidig utvikling

Walkthrough / gjennomgang

- Hovedmål: Finne feil, forbedre programvareproduktet, analysere alternative løsninger, evaluere samsvar med standarder og spesifikasjoner
- Mulige tilleggsmål: Utveksle ideer om variasjoner i teknikk og stil, lære opp deltakerne, oppnå konsensus
- Individuell forberedelse før reviewmøtet er ikke påkrevd
- Reviewmøtet blir typisk ledet av forfatteren eller utvikler
- Det skal være en protokollfører
- Bruk av sjekklister er valgfritt

- Kan ta formen av scenarier, prøvekjøringer eller simuleringer
- Feillogger og reviewrapporter kan lages
- Kan i praksis variere fra meget uformelt til meget formelt

Teknisk review

- Hovedmål: Oppnå konsensus, finne mulige feil
- Mulige tilleggsmål: Evaluere kvalitet og bygge tillit til arbeidsproduktet. Generere nye ideer, motivere forfattere og gjøre dem i stand til å forbedre framtidige arbeidsprodukter. Undersøke alternative løsninger
- Reviewere bør være på teknisk samme nivå med forfatteren og tekniske eksperter i den samme eller andre yrkesretninger
- Individuell forberedelse før reviewmøtet er påkrevd
- Reviewmøtet er valgfri. Der det finner sted blir det i beste fall ledet av en opplært moderator (typisk ikke forfatteren)
- Det skal være en protokollfører, ideelt er det ikke forfatteren
- Bruk av sjekklister er valgfritt
- Feillogger og reviewrapporter blir vanligvis laget

Inspeksjon

- Hovedmål: Finne mulige feil, evaluere kvalitet og bygge tillit til arbeidsproduktet. Unngå framtidige liknende feil ved feilkildeanalyse og ved at forfatteren lærer
- Mulige tilleggsmål: å motivere forfattere og gjøre de i stand til å forbedre framtidige arbeidsprodukter og utviklingsprosessen for programvare, oppnå konsensus
- Følger en definert prosess med formelt dokumenterte resultater, basert på regler og sjekklister
- En bruker klart definerte roller slik de er spesifisert i avsnitt 3.2.2. Disse er påkrevd. Rollene kan også omfatte en spesielt utvalgt leser som leser opp arbeidsproduktet under møtet
- Individuell forberedelse før reviewmøtet er påkrevd
- Reviewere er enten på teknisk samme nivå som forfatteren (eng: peers) eller eksperter i andre fagområder som er relevante for arbeidsproduktet
- Det brukes spesifiserte start- og sluttkriterier
- En protokollfører er påkrevd
- Reviewmøtet ledes av en opplært moderator (ikke forfatteren)
- Forfatter kan ikke være reviewleder, leser eller protokollfører
- Det lages logger av mulige feil og en reviewrapport
- Måledata samles og brukes til å forbedre hele prosessen for programvareutvikling og selve inspeksjonsprosessen

Et enkelt arbeidsprodukt kan utsettes for flere typer review. Hvis flere reviews blir brukt kan rekkefølgen variere. F.eks. kan en uformell review gjøres før en teknisk review for å sikre at arbeidsproduktet er klart til teknisk review.

Reviewtypene som er beskrevet ovenfor kan utføres som såkalte «peer reviews», også kalt fagfellevurderinger, dvs. de blir utført av kolleger kvalifisert til å gjøre det samme arbeidet.

Typene feil en finner i en review varierer. De avhenger spesielt av arbeidsproduktet under review. Se avsnitt 3.1.3 for eksempler på feil som kan finnes i ulike arbeidsprodukter, se også Gilb 1993 for mer informasjon om inspeksjoner.

3.2.4 Bruk av reviewteknikker

Det finnes ulike reviewteknikker til bruk under individuell review (individuell forberedelse) for å finne feil. Disse teknikkene kan brukes i alle reviewtypene ovenfor. Teknikkenes effektivitet vil variere avhengig av reviewtypen hvor disse blir brukt. Eksempler for ulike teknikker for individuell review er listet nedenfor.

Ad hoc

I en ad hoc review har reviewere ingen eller lite veiledning for hvordan oppgaven skal utføres. Ofte leser reviewere arbeidsproduktet sekvensielt og identifiserer og dokumenterer ting de finner underveis. Ad hoc review er en ofte benyttet teknikk som krever lite forberedelse. Denne teknikken avhenger meget av reviewerens evner og kan føre til at mange ting blir meldt flere ganger hvis en bruker flere reviewere.

Sjekklistebasert

Sjekklistebasert review er en systematisk teknikk der reviewere finner ting basert på sjekklister. Disse blir utdelt ved invitasjonen til en review (f.eks. av reviewlederen). En review-sjekkliste består av et antall spørsmål basert på typiske feil, som kan være utledet av erfaring. Sjekklister bør være spesielt laget for hver type arbeidsprodukt under review og bør vedlikeholdes regelmessig for å dekke problemer en har oversett i tidligere reviews. Hovedfordelen til den sjekklistebaserte teknikken er at en systematisk evaluerer produktet for typiske feil. En bør være forsiktig, slik at en ikke bare følger sjekklisten. En bør også se etter feil som ikke er nevnt i sjekklisten.

Scenarier og tørrkjøringer

I en scenariobasert review får reviewerne strukturerte retningslinjer om hvordan de skal lese gjennom arbeidsproduktet. En scenariobasert tilnærming understøtter reviewere i å utføre såkalte “dry runs”, tørrkjøringer, på arbeidsproduktet basert på forventet bruk av arbeidsproduktet (hvis arbeidsproduktet er dokumentert i et passende format som f.eks. brukstilfeller). Disse scenariene gir reviewere bedre retningslinjer for å identifisere spesifikke feiltyper enn enkle sjekklistepunkter. Som med sjekklistebasert reviewteknikk bør reviewere ikke føle seg begrenset til de dokumenterte scenariene, men også se utenfor dem for å finne andre feil (f.eks. funksjoner som mangler).

Rollebasert

Rollebasert review er en teknikk der en evaluerer arbeidsproduktet fra perspektivene til individuelle interessenter eller roller. Typiske roller omfatter spesifikke type brukere (erfarne, uerfarne, gamle, barn osv.) samt spesifikke roller i organisasjonen (brukeradministrator, systemadministrator, ytelsestester, sikkerhetsansvarlig osv.).

Perspektivbasert

I perspektivbasert lesing ivaretar reviewere synspunkter til forskjelle interessenter, når de individuelt gjennomgår arbeidsproduktet. Dette ligner på rollebasert review. Typiske interessentsynspunkter omfatter sluttbruker, markedsføring, designer, tester eller driftsansvarlig. (Merk forskjellen fra rollebasert review der en varierer den enkelte interessenten). Å bruke forskjellige interessentsynspunkter fører til mer dybde (grundighet) under individuell review med mindre duplisering av problemer på tvers av reviewere.

I tillegg krever perspektivbasert lesing også at reviewere prøver å bruke arbeidsproduktet under review for å lage produktet som de vil lage med utgangspunkt i arbeidsproduktet. F.eks. ville en tester prøve å lage utkast til akseptansetester hvis han/hun utfører perspektivbasert lesing på en kravspesifikasjon for å

se om all nødvendig informasjon er med. I perspektivbasert lesing forventer en i tillegg at sjekklister brukes.

Empiriske studier har vist at perspektivbasert lesing er den mest effektive generelle teknikken for å gjennomgå krav og tekniske arbeidsprodukter. En nøkkelfaktor for suksess er å inkludere og vekte ulike interessentsynspunkter fornuftig, basert på risiko. Se Shul 2000 for detaljer om perspektivbasert lesing, og Sauer 2000 for effektiviteten av ulike reviewtyper.

3.2.5 Suksessfaktorer for reviews

For å få en suksessfull review må en vurdere rett reviewtype og reviewteknikker. I tillegg finnes en del andre faktorer som har innflytelse på resultatet av en review.

Organisatoriske suksessfaktorer for reviews omfatter:

- Hver review har klare mål som blir definert under reviewplanlegging og senere brukt som målbare sluttkriterier
- Det brukes egnede reviewtyper. De skal være egnet til å oppnå målsetningene og skal være passende til type og nivå av programvare-arbeidsproduktene som blir gjennomgått og deltakerne
- Deltakerne må ha nok tid til å forberede seg
- Deltakerne må ha nok ekspertise
- Reviews blir planlagt tidlig nok
- Enhver reviewteknikk som brukes, som sjekkliste- eller rollebasert review, må være egnet for effektiv feilfinning i arbeidsproduktet som blir gjennomgått
- Sjekklistene må nevne de vesentlige risikofaktorer og være oppdatert
- Store dokumenter skrives og blir gjennomgått i mindre deler, slik at kvalitetskontrollen blir gjort ved å gi forfatterne tidlig og hyppig tilbakemelding om feil
- Ledelsen støtter reviewprosessen, f.eks. ved å sette av nok tid for reviewaktiviteter i prosjektplaner

Personrelaterte suksessfaktorer for reviews omfatter:

- Deltakerne bruker nok tid og er grundige
- De rette menneskene blir involvert for å oppnå målsettingene med review. F.eks. brukes mennesker med ulike evner og perspektiver og personer som skal kunne bruke dokumentet som arbeidsinput
- Testere blir ansett som verdifulle bidragsytere til review. Ved å delta lærer de om arbeidsproduktet, noe som tillater dem å forberede mer effektive tester og å lage disse tidligere
- Reviews blir utført på mindre deler eller avsnitt, slikt at reviewere ikke mister konsentrasjonen under den individuelle forberedelsen og/eller under reviewmøtet (hvis det holdes)
- Feil som finnes blir anerkjent og håndtert objektivt
- Møtet ledes på en god måte, slik at deltakerne anser den som verdifull bruk av deres tid
- Selve review utføres i et miljø preget av tillit. Resultatet brukes ikke til å evaluere deltakerne
- Deltakerne viser god folkeskikk
- Det gis passende opplæring, spesielt for de mer formelle reviewtyper som inspeksjoner

- Det praktiseres en kultur for læring og prosessforbedring

Se Gilb 1993, Wiegers 2002 og van Veenendaal 2004 for mer informasjon om suksessfaktorer for reviews.

4 Testteknikker — 330 minutter

Nøkkelord

beslutningsdekning, beslutningstabelltesting, ekvivalensklasseinndeling, erfaringsbasert testteknikk, feilgjetting, grenseverdianalyse, hvitboks testteknikk, programinstruksjonsdekning, sjekklistebasert testing, svartboks testteknikk, testdekning, testteknikk, tilstandsbasert testing, testing basert på brukstilfelle (use case testing), utforskende testing

Læremål

4.1 Kategorier av testteknikker

FL-4.1.1 (K2) Forklar egenskapene og hva som er felles hhv. forskjellig mellom svartboks testteknikker, hvitboks testteknikker og erfaringsbaserte testteknikker

4.2 Svartboks testteknikker

FL-4.2.1 (K3) Bruk ekvivalensklasseinndeling for å utlede testtilfelle fra gitte krav

FL-4.2.2 (K3) Bruk grenseverdianalyse for å utlede testtilfelle fra gitte krav

FL-4.2.3 (K3) Bruk beslutningstabelltesting for å utlede testtilfelle fra gitte krav

FL-4.2.4 (K3) Bruk tilstandsbasert testing for å utlede testtilfelle fra gitte krav

FL-4.2.5 (K2) Forklar hvordan en utleder testtilfelle fra et brukstilfelle

4.3 Hvitboks testteknikker

FL-4.3.1 (K2) Forklar programinstruksjonsdekning

FL-4.3.2 (K2) Forklar beslutningsdekning

FL-4.3.3 (K2) Forklar verdien av programinstruksjonsdekning og beslutnings- eller forgreningsdekning

4.4 Erfaringsbaserte testteknikker

FL-4.4.1 (K2) Forklar feilgjetting

FL-4.4.2 (K2) Forklar utforskende testing

FL-4.4.3 (K2) Forklar sjekklistebasert testing

4.1 Kategorier av testteknikker

Formålet med testteknikker, også de som behandles i dette avsnittet, er å hjelpe med å identifisere testbetingelser, testtilfelle og testdata.

4.1.1 Valg av testteknikker

Valget hvilke testteknikker en skal bruke, avhenger av mange faktorer, bl.a. følgende:

- Type og kompleksitet av komponenten eller systemet
- Standarder og regelverk fra myndighetene
- Krav fra kunde eller kontrakt
- Risikonivåer og risikotyper
- Målet med testen
- Tilgjengelig dokumentasjon
- Testernes kunnskap, evner og erfaring
- Tilgjengelige verktøy
- Tid og budsjett
- Utviklingsmodellen
- Forventet bruk
- Tidligere erfaringer med å bruke testteknikkene på komponenten eller systemet som skal testes
- Type feil en forventer i komponenten eller systemet

Noen teknikker er mer anvendelige i noen situasjoner og for noen testnivåer; andre er brukbare for alle testnivåer. For å nå et best mulig resultat, bruker testere generelt en kombinasjon av testteknikker når de lager testtilfelle.

Bruken av testteknikker i testanalyse, testdesign og testimplementering kan variere fra uformell (med lite eller ingen dokumentasjon) til veldig formell. Det rette nivå av formalitet avhenger av omstendighetene rundt testingen, bl.a. modenheten til test- og utviklingsprosessene, tidsbegrensninger, sikkerhetskrav, myndighetskrav, kunnskap og evne til de involverte personene og utviklingsmodellen.

4.1.2 Kategorier av testteknikker og deres kjennetegn

I dette pensum blir testteknikkene klassifisert som svartboks, hvitboks eller erfaringsbasert.

Svartboks testteknikker (også kalt oppførselsbaserte teknikker) baserer seg på en analyse av testgrunnlaget (f.eks. formelle kravdokumenter, spesifikasjoner, brukerhistorier og brukstilfeller, arbeidsrutiner eller forretningsprosesser). Disse teknikkene kan brukes for både funksjonell og ikke-funksjonell testing. Svartboks testteknikker konsentrerer seg om input til og output fra testobjektet, uten referanse til dets interne struktur.

Hvitboks testteknikker (også kalt strukturelle eller strukturbaserte teknikker) baserer seg på en analyse av arkitekturen, detaljdesignet, den interne strukturen eller koden til testobjektet. I motsetning til svartboks testteknikker konsentrerer hvitboks testteknikker seg på strukturen og prosessforløpet i testobjektet.

Erfaringsbaserte testteknikker bruker erfaringene til utviklere, testere og brukere for å konstruere, implementere og utføre tester. Disse teknikkene blir ofte kombinert med svartboks og hvitboks testteknikker.

Svartboks testteknikker har bl.a. følgende felles kjennetegn:

- Testbetingelser, testtilfelle og testdata blir avledet fra et testgrunnlag som kan omfatte krav til programvaren, spesifikasjoner, brukstilfeller og brukerhistorier
- Testtilfelle kan brukes for å finne avvik mellom kravene og deres implementasjon, pluss avvik fra kravene
- Testdekning blir målt basert på det som er nevnt i testgrunnlaget og testteknikken som anvendes på testgrunnlaget.

Hvitboks testteknikker har bl.a. følgende felles kjennetegn:

- Testbetingelser, testtilfelle og testdata blir avledet fra et testgrunnlag som kan omfatte kode, programvarearkitektur, detaljdesign eller hvilken som helst annen informasjonskilde om programvarens struktur
- Testdekning blir målt basert på strukturelementer (f.eks. kode eller grensesnitt)
- Spesifikasjoner blir ofte brukt som en ekstra informasjonskilde for å kunne bestemme hvilke resultater som forventes av testtilfellene

Erfaringsbaserte testteknikker har bl.a. følgende felles kjennetegn:

- Testbetingelser, testtilfelle og testdata blir avledet fra et testgrunnlag som kan omfatte kunnskap og erfaring til testere, utviklere, brukere og andre interessenter

Denne kunnskapen og erfaringen kan gjelde forventet bruk av programvaren, dens omgivelse, mulige feil og deres fordeling.

Den internasjonale standarden (ISO/IEC/IEEE 29119-4) inneholder beskrivelser av testteknikker og deres tilsvarende måter å måle testdekningen på (se Craig 2002 og Copeland 2004 for mer om teknikker).

4.2 Svartboks testteknikker

4.2.1 Ekvivalensklasseinndeling

Ekvivalensklasseinndeling deler opp data in delmengder (også kjent som ekvivalensklasser) slik at alle medlemmer i en delmengde forventes behandlet på samme måte (se Kaner 2013 og Jørgensen 2014). Det finnes ekvivalensklasser for både gyldige og ugyldige verdier.

- Gyldige verdier er verdier som bør godtas av komponenten eller systemet. En ekvivalensklasse som inneholder gyldige verdier blir kalt “gyldig ekvivalensklasse”.
- Ugyldige verdier er verdier som bør avvises av komponenten eller systemet. En ekvivalensklasse som inneholder ugyldige verdier blir kalt “ugyldig ekvivalensklasse”.
- Ekvivalensklasser kan identifiseres for hvilket som helst dataelement som hører til testobjektet, inklusive inputdata, outputdata, interne verdier, tidsrelaterte verdier, f.eks. før og etter en hendelse og for grensesnittparametere, f.eks. testing av integrerte komponenter under integrasjonstesting.
- Alle ekvivalensklasser kan ved behov deles opp i underklasser.

- Hver verdi må tilhøre en (og kun en) ekvivalensklasse.
- Når ugyldige ekvivalensklasser brukes i testtilfelle, bør disse testes en og en, dvs. de skal ikke kombineres med andre ugyldige ekvivalensklasser. Dette anbefales for å sikre at ikke feil blir skjult og dermed oversett. Hvis det er flere feil samtidig, men bare en av dem fører til et synlig (feil) resultat, blir de andre feilene skjult. De blir da ikke funnet.

For å oppnå 100% testdekning med denne teknikken, må testtilfellene dekke alle de identifiserte klassene (også de ugyldige) ved å bruke minst en verdi fra hver klasse. Testdekningen måles som antall klasser testet med minst en verdi dividert med det totale antall identifiserte ekvivalensklasser, vanligvis som et prosenttall.

Ekvivalensklasseinndeling kan brukes på alle testnivåer.

4.2.2 Grenseverdianalyse

Grenseverdianalyse er en utvidelse av ekvivalensklasseinndeling, men kan bare brukes når klassen består av verdier i en rekkefølge, dvs. den inneholder tall eller sekvensielle data. De minste og største verdiene (eller første og siste) av en klasse er dens grenseverdier (Beizer 1990).

F.eks. anta at et inputfelt godtar et enkelt siffer som input, og at en bruker et tastatur med kun sifre. Det gyldige område skal være fra og med 1 til og med 5. Altså har en tre ekvivalensklasser: ugyldig (lavere enn 1), gyldig (fra og med 1 til og med 5) og ugyldig (høyere enn 5). For den gyldige ekvivalensklassen er grenseverdiene 1 og 5. For den ugyldige klassen (for høyt) er grenseverdiene 6 og 9. For den andre ugyldige klassen (for lavt) finnes bare en grenseverdi, 0, fordi klassen har bare et medlem, nemlig 0.

I dette eksempelet identifiserer vi to grenseverdier per grense. Grensen mellom ugyldig (for lavt) og gyldig gir testverdiene 0 og 1. Tilsvarende gir grensen mellom gyldig og ugyldig (for høyt) testverdiene 5 og 6. En variant av denne teknikken identifiserer tre verdier per grense: verdiene før, på og akkurat over grensen. I det gitte eksempelet vil det resultere i verdiene 0, 1 og 2 for den nedre grensen og verdiene 4, 5 og 6 for den øvre grensen. (Jørgensen 2014).

Oppførselen ved grensene til ekvivalensklasser har større sannsynlighet til å være feil enn oppførsel lenger inne i klassene. Det er viktig å huske at både spesifiserte og implementerte grenser kan være forskjøvet til (feile) posisjoner over eller under deres ønskede posisjoner, at de kan være helt utelatt, eller at de kan være erstattet av uønskede ekstra grenser. Grenseverdianalyse og -testing avslører nesten alle slike feil ved å tvinge programvaren å vise oppførsel fra en annen ekvivalensklasse enn den grenseverdien egentlig skal høre til.

Grenseverdianalyse kan brukes på alle testnivåer. Denne teknikken brukes generelt til å teste krav i forhold til tallområder (også datoer og tider). Grenseverdidekning for en klasse måles som antall grenseverdier testet, dividert med det totale antall identifiserte grenseverdier, vanligvis som et prosenttall.

4.2.3 Beslutningstabelltesting

Testteknikker for kombinasjoner er nyttige for å teste implementasjonen av systemkrav som spesifiserer hvordan ulike kombinasjoner av betingelser skal gi ulike resultater. En måte å teste dette på er beslutningstabelltesting.

Beslutningstabeller er en god måte å beskrive kompliserte forretningsregler som et system må følge. Når testeren setter opp beslutningstabeller, identifiserer hun betingelser (ofte inndata) og resulterende aksjoner (ofte utdata). Betingelsene og resultatene skrives under hverandre i en tabell, vanligvis i venstre kolonne, vanligvis med betingelsene øverst og aksjonene nederst. Hver kolonne i resten av tabellen blir utfylt med en kombinasjon av betingelser. Den tilsvarer dermed en beslutningsregel som definerer en unik kombinasjon av betingelser som resulterer i utføring av aksjoner for denne regelen. Verdiene av disse betingelsene og aksjonene blir vanligvis vist som sannhetsverdier (sann, feil) eller diskrete verdier

(f.eks. rødt, grønt, hvit, …). Disse forskjellige typer av betingelser og aksjoner kan forekomme i samme tabell.

Den vanlige notasjonen er som følger:

For betingelser:

- Y betyr at betingelsen er sann (kan også vises som S eller 1)
- N betyr at betingelsen er feil (kan også vises som F eller 0)
- En strek betyr at betingelsen spiller ingen rolle (kan også vises som N/A)

For aksjoner:

- X betyr at aksjonen skal skje (kan også vises som J eller S eller 1)
- 'Tomt felt' betyr at aksjonen ikke skal skje (kan også vises som N eller F eller strek eller 0)

En fullstendig beslutningstabell har nok kolonner til å dekke hver kombinasjon av betingelser. Tabellen kan reduseres ved å slette kolonner som inneholder umulige kombinasjoner av betingelser, kolonner som inneholder mulige men ikke oppnåelige kombinasjoner og kolonner som tester kombinasjoner som ikke har effekt på resultatet. For mer informasjon om hvordan en kan redusere beslutningstabeller, se ISTQB Advanced Level Test Analyst Syllabus - ATA.

Minimal testdekning for beslutningstabelltesting er vanligvis å ha minst ett testtilfelle per beslutningsregel i tabellen. Dette betyr at man vanligvis dekker alle kombinasjoner av betingelser, altså alle kolonner i tabellen. Testdekningen måles ved å dividere antall beslutningsregler som er testet med minst ett testtilfelle med det totale antallet beslutningsregler. Vanligvis uttrykker man testdekningen i prosent.

Styrken til beslutningstabelltesting er at teknikken hjelper å identifisere alle viktige kombinasjoner av betingelser. Noen av dem kan ellers bli oversett. Metoden finner også "hull" i kravene. Metoden kan brukes i alle situasjoner der programvarens oppførsel er avhengig av en kombinasjon av betingelser. Teknikken kan brukes på alle testnivåer.

4.2.4 Tilstandsbasert testing

Komponenter eller systemer kan reagere ulikt på en hendelse, avhengig av nåværende betingelser eller historien før (f.eks. hendelsene som har skjedd siden systemet ble startet). Den tidligere historien kan oppsummeres ved å bruke konseptet "tilstand". Et tilstandsovergangsdiagram eller tilstandsdiagram viser de mulige tilstandene til programvaren og hvordan programvaren går inn i en tilstand, går ut av en tilstand eller går fra en tilstand over til en annen. En overgang blir trigget av en hendelse (f.eks. brukerinput av en verdi i et felt). Hendelsen resulterer i en overgang. Hvis den samme hendelsen kan resultere i to eller flere ulike overganger fra den samme tilstanden, kan hendelsen bli utstyrt med en "guard"-betingelse. Tilstandsendringen kan resultere i at programvaren gjør en aksjon (f.eks. at den skriver ut et beregningsresultat eller en feilmelding).

OBS: «Guard» betingelser ignoreres i tilstandsbasert testing og testes med andre teknikker.

En tilstandsovergangstabell viser alle gyldige og potensielt ugyldige overganger mellom tilstander. I tillegg viser den hendelser, "guard"-betingelser og resulterende aksjoner for gyldige overganger. Tilstandsdiagrammer viser vanligvis bare de gyldige overgangene og utelukker de ugyldige.

En kan lage tester som dekker en typisk følge av tilstander, alle tilstander, hver overgang, spesielle følger av overganger eller ugyldige overganger.

Tilstandsbasert testing brukes for menybaserte applikasjoner og er mye brukt til test av «embedded» systemer. Teknikken egner seg også for å modellere et forretningsscenario som har spesielle tilstander

eller for å teste navigeringen på skjermen. 'Tilstand' er et abstrakt konsept – det kan representere noen få kodelinjer eller en hel forretningsprosess.

Testdekning blir vanligvis målt ved å dividere antall identifiserte tilstander eller overganger som er testet med det totale antall tilstander eller overganger i testobjektet. Vanligvis uttrykkes testdekningen som et prosenttall. For mer informasjon om dekningskriterier for tilstandsbasert testing, se ISTQB Advanced Level Test Analyst Syllabus (ATA).

4.2.5 Testing basert på brukstilfelle (Use Case Testing)

En kan utlede tester fra brukstilfeller. Dette er en måte å konstruere interaksjoner med programvare. En tester krav for funksjonene som disse brukstilfellene representerer. Brukstilfeller er forbundet med aktører (menneskelige brukere, ekstern hardware, eller andre komponenter eller systemer) og subjekter (komponenten eller systemet som et brukstilfelle går imot).

Hvert brukstilfelle spesifiserer en oppførsel som testobjektet kan utføre i samarbeid med en eller flere aktører (UML 2.5.1 2017). Et brukstilfelle kan beskrives ved hjelp av interaksjoner og aktiviteter, i tillegg til forutsetninger, etterbetingelser og naturlig språk hvor det er passende. Interaksjoner mellom aktørene og subjektet kan resultere i tilstandsendringer for subjektet.

Interaksjoner kan vises grafisk ved hjelp av arbeidsflyt, aktivitetsdiagrammer eller modeller av forretningsflyt.

Et brukstilfelle består vanligvis av en hovedflyt og variasjoner av denne. Eksempler er unntakshåndtering og behandling av feilsituasjoner (systemsvar og gjenoppretting fra programmeringsfeil, kommunikasjonsfeil og brukerfeil som f.eks. resulterer i en feilmelding).

Tester blir konstruert for å utføre de definerte oppførslene (hoved, unntak, alternative og feilhåndtering). Testdekning kan måles ved å dividere antall testede flyt med det totale antall flyt, vanligvis uttrykt i prosent.

Mer informasjon om dekningskriterier for testing basert på brukstilfelle, se ISTQB Advanced Level Test Analyst Syllabus - ATA.

4.3 Hvitboks testteknikker

Hvitboks testing baserer seg på den interne strukturen til testobjektet. Hvitboks testteknikker kan brukes på alle testnivåer, men de to koderelaterte teknikkene som beskrives i dette avsnittet brukes vanligvis i komponenttesten. Det finnes mer avanserte teknikker som brukes i noen sikkerhetskritiske og forretningskritiske omgivelser eller der det kreves høy integritet. Disse diskuteres ikke her. For mer informasjon om disse teknikkene, se ISTQB Advanced Level Technical Test Analyst syllabus - ATA.

4.3.1 Programinstruksjonstesting og -dekning

Programinstruksjonstesting tester de utførbare programinstruksjonene i koden. Testdekning måles som antall programinstruksjoner utført av testene dividert med det totale antall utførbare programinstruksjoner i testobjektet, vanligvis uttrykt i prosent.

4.3.2 Beslutningstesting og -dekning

Beslutningstesting tester beslutningene i koden og koden som blir utført basert på resultatene av disse beslutningene. For å gjøre dette, følger testtilfellene kontrollflytene som går ut fra et beslutningspunkt, f.eks. for en IF-instruksjon, en for det sanne resultat og en for det feile; for en CASE-instruksjon kreves testtilfelle for hvert spesifisert resultat, pluss for standardresultatet.

Testdekning måles som antall beslutningsresultater som er utført av testene dividert med det totale antall beslutningsresultater i testobjektet, vanligvis uttrykt i prosent.

4.3.3 Verdien av programinstruksjons- og beslutningstesting

Når 100% programinstruksjonsdekning er oppnådd, sikrer det at alle utførbare instruksjoner i koden har blitt testet minst en gang. Men det sikrer ikke at all beslutningslogikk er testet. Programinstruksjonsdekning kan gi lavere testdekning enn beslutningsdekning.

Når 100% beslutningsdekning er oppnådd, sikrer det at alle beslutningsresultater er utført. Dette omfatter testing av både rett og feil resultat av beslutningene, selv om det av og til ikke finnes en eksplisitt instruksjon for feil resultat (f.eks. når en IF-instruksjon ikke har en ELSE i koden). Beslutningsdekning hjelper til å finne feil i kode der andre tester ikke har utført både rett og feil resultat.

Å oppnå 100% beslutningsdekning garanterer 100% programinstruksjonsdekning (men ikke omvendt).

4.4 Erfaringsbaserte testteknikker

Når en bruker erfaringsbaserte testteknikker, utvikler en testtilfellene fra testernes evner og intuisjon og deres erfaring med liknende applikasjoner og teknologier. Disse teknikkene kan hjelpe til å identifisere tester som ikke er så lett å identifisere med andre mer systematiske teknikker. Avhengig av testerens framgangsmøte og erfaring, kan disse teknikkene oppnå veldig varierende grader av testdekning og effektivitet. Testdekning kan være vanskelig å bedømme og kan muligens ikke måles med disse teknikkene.

Vanligvis brukte erfaringsbaserte teknikker blir diskutert i de følgende avsnittene.

4.4.1 Feilgjetting

Feilgjetting er en teknikk som brukes for å forutsi forekomsten av feil. Den er basert på testerens kunnskap, bl.a.:

- Hvordan applikasjonen har virket før
- Hva type feil utviklerne gjør
- Feil som har skjedd i andre applikasjoner

En metodisk framgangsmåte for feilgjetting er å lage en liste over mulige feil og så konstruere tester som vil avdekke disse feil og deres årsaker. Disse feillister kan en bygge basert på erfaring, data om feil eller fra generell kunnskap om hvorfor programvare feiler.

4.4.2 Utforskende testing (eng: exploratory testing)

I utforskende testing blir uformelle tester konstruert, utført, loggført og evaluert dynamisk under testutføringen. Testresultatene blir brukt for å lære mer om komponenten eller systemet og for å lage tester for de områdene som må testes mer.

Utforskende testing blir av og til utført ved å bruke sesjonsbasert testing. Dette gjøres for å strukturere aktiviteten. Der utfører en utforskende testing i et forhåndsdefinert tidsintervall. Testeren bruker da et testcharter som inneholder testmål for å veilede testingen. Testeren kan dokumentere stegene som er gjort og hva en har observert.

Utforskende testing er mest nyttig når en har lite eller dårlige spesifikasjoner eller sterkt tidspress for testingen. Utforskende testing er også nyttig til å utfylle andre mer formelle testteknikker.

Utforskende testing henger sterkt sammen med reaktive teststrategier (se avsnitt 5.2.2). Utforskende testing kan anvende andre svartboks, hvitboks og erfaringsbaserte teknikker.

4.4.3 Sjekklistebasert testing

I sjekklistebasert testing konstruerer, implementerer og utfører testerne tester for å dekke testbetingelser funnet i en sjekkliste. Som del av analysen velger testerne en eksisterende sjekkliste, oppdaterer den eller lager en ny sjekkliste. Slike sjekklister kan lages basert på erfaring, kunnskap om hva som er viktig for brukeren, eller en forståelse for hvorfor og hvordan programvare feiler.

Sjekklister kan lages for å støtte forskjellige testtyper, bl.a. funksjonell og ikke-funksjonell testing. Når en ikke har detaljerte testtilfelle, kan sjekklistebasert testing gi retningslinjer og en viss grad av konsistens. Fordi sjekklistene er på høyt nivå, vil testingen variere en del. Dette resulterer potensielt i større testdekning men mindre mulighet å gjenta testene.

5 Testledelse — 225 minutter

Nøkkelord

feilhåndtering, feilrapport, framdriftsrapport for test, konfigurasjonsstyring, produktrisiko, prosjektrisiko, risiko, risikobasert testing, risikonivå, sluttkriterier, sluttrapport for test, startkriterier, tester, testestimering, testleder, testovervåking, testplan, testplanlegging, teststrategi, teststyring, testtilnærming

Læremål

5.1 Testorganisasjon

FL-5.1.1 (K2) Forklar fordelene og ulempene ved uavhengig testing

FL-5.1.2 (K1) Identifiser oppgavene til testleder og til tester

5.2 Testplanlegging og -estimering

FL-5.2.1 (K2) Oppsummer hensikten med testingen og innholdet til en testplan

FL-5.2.2 (K2) Forklar forskjellen mellom ulike teststrategier

FL-5.2.3 (K2) Gi eksempler på start- og sluttkriterier

FL-5.2.4 (K3) Anvend kompetanse om prioritering og tekniske og logiske avhengigheter for å sette opp en kjøreplan for en gitt mengde testtilfeller

FL-5.2.5 (K1) Identifiser faktorer som påvirker testinnsatsen

FL-5.2.6 (K2) Forklar forskjellen mellom de to estimeringsteknikkene: statistikkbasert og ekspertbasert

5.3 Testovervåking og -styring

FL-5.3.1 (K1) List opp hvordan testing kan måles

FL-5.3.2 (K2) Oppsummer hensikten med testingen, innhold og målgrupper for testrapporter

5.4 Konfigurasjonsstyring

FL-5.4.1 (K2) Oppsummer hvordan konfigurasjonsstyring støtter testing

5.5 Risikoer og testing

FL-5.5.1 (K1) Definer risikonivå ved å bruke sannsynlighet og konsekvens

FL-5.5.2 (K2) Forklar forskjellen mellom prosjekt- og produktrisikoer

FL-5.5.3 (K2) Beskriv ved å bruke eksempler hvordan analyse av produktrisiko kan påvirke testgrundigheten og testomfanget

5.6 Feilhåndtering

FL-5.6.1 (K3) Skriv en feilrapport om hendelser funnet under testing

5.1 Testorganisasjon

5.1.1 Uavhengig testing

Testoppgaver kan utføres av testprofesjonelle eller av personer i andre roller, f.eks. kunder. En viss grad av uavhengighet gjør ofte at testeren blir mer effektiv i å finne feil pga. at testeren ikke er forutinntatt sammenliknet med forfatterens egen evne til å finne feil i sitt produkt ("cognitive biases", se avsnitt 1.5). Uavhengighet er riktignok ikke en erstatning for kjennskap til produktet, og utviklere kan raskt og effektivt finne mange feil i sine egne kodelinjer, heretter forkortet koden.

Uavhengighet i testing omfatter, sortert fra lav til høy grad av uavhengighet:

- Ingen uavhengige testere, dvs. den eneste testingen er at utviklere tester sin egen kode
- Uavhengige utviklere eller testere innen utviklingsteamene eller prosjektteamet, dette kan være utviklere som tester sine kollegers produkter
- Uavhengige testteam eller en gruppe i organisasjonen som rapporterer direkte til prosjektleder eller linjeleder
- Uavhengige testere fra forretningssiden av organisasjonen. Dette kan også være brukergrupper eller testere som er spesialisert i ulike testtyper slik som brukbarhet, sikkerhet, ytelse, forskriftskrav eller portabilitet
- Uavhengige testere utenfor organisasjonen, enten på arbeidsplassen (innleid) eller utenfor arbeidsplassen («outsourcing»)

For de fleste typer prosjekter er det vanligvis best å ha flere testnivåer der noen av disse nivåene håndteres av uavhengige testere. Utviklere bør delta i testing, spesielt på de lavere nivåer, for å ta kontroll over kvaliteten på sitt eget arbeid.

Måten uavhengighet i testingen gjennomføres på, varierer avhengig av utviklingsmodellen. I smidig (agile) utvikling kan f.eks. testerne være del av et utviklingsteam. I noen organisasjoner som bruker smidige metoder, kan disse testerne like gjerne betraktes som del av et større, uavhengig testteam. I slike organisasjoner kan produkteiere i tillegg utføre akseptansetesting for å validere brukerhistorier ved slutten av hver iterasjon.

Mulige fordeler ved uavhengighet i testingen:

- Uavhengige testere finner sannsynligvis andre typer feil enn utviklere pga. at de har ulik bakgrunn, ulik teknisk synsvinkel og mht. utfordringen med forutinntatthet
- En uavhengig tester kan verifisere, utfordre eller motbevise interessentenes antakelser i spesifikasjonen og byggingen av systemet.

Mulige ulemper ved uavhengighet i testingen:

- Testerne er isolert fra utviklingsteamet. Noe som kan lede til mangel på samarbeid, forsinkelser i tilbakemeldingene og at teamene kan bli i opposisjon til hverandre.
- Utviklere kan tape ansvarsfølelse i forhold til kvaliteten
- Uavhengige testere kan bli sett på som flaskehals eller anklaget for å forsinke leveransen
- Uavhengige testere kan mangle noe viktig informasjon f.eks. om testobjektet

Mange organisasjoner er i stand til å oppnå fordelene med uavhengig testing samtidig som de unngår ulempene.

5.1.2 Testleders og testers oppgaver

I dette pensum dekkes to roller i forhold til test: testleder og tester. Aktivitetene og oppgavene som utføres av disse to rollene avhenger av prosjektet, produktet, organisasjonen og kompetansen til de som har rollene.

Testleder har hovedansvaret for testprosessen og for å lede testaktivitetene med suksess. Testlederrollen kan utføres av en profesjonell testleder, prosjektleder, utviklingsleder eller en kvalitetsleder. I større prosjekter eller organisasjoner kan det være flere testteam som rapporterer til en testleder, testrådgiver eller testkoordinator. Hvert av disse testteam kan ha en teamleder som rapporterer til testleder.

Typiske testlederoppgaver:

- Lage eller granske organisasjonens testpolicy og teststrategi
- Planlegge testaktivitetene på basis av testleders analyse av kontekst til testen, testmålene og risikobildet for testen. Dette kan omfatte valg av testtilnærminger, estimering av tiden som kreves for testingen, testinnsats og kostnad, bestilling av ressurser, definisjon av testnivåer og testsykluser og planlegging av feilhåndtering
- Skrive og oppdatere testplan(er)
- Koordinere testplan(er) med prosjektledere, produkteiere og andre relevante roller
- Informere om testperspektivet i forhold til andre prosjektaktiviteter slik som integrasjonsplanleggingen
- Initiere analyse, design, bygging og utføring av tester, overvåke testframdriften og testresultatene og kontrollere statusen til sluttkriteriene (eller definisjonen av "done")
- Forberede og levere framdriftsrapporter og sluttrapport for test på basis av innsamlet informasjon
- Justere planene basert på testresultater og testframdrift (noen ganger dokumentert i framdriftsrapporter for testen og/eller i sluttrapporter fra annen, fullført testing i prosjektet) og utøve teststyring i form av å iverksette nødvendige tiltak
- Delta i å sette opp prosess for feilhåndtering og tilstrekkelig konfigurasjonsstyring av testmateriell
- Sette opp måleparametere for testframdrift og for å vurdere kvaliteten både til testingen og til produktet
- Delta i å velge og ta i bruk verktøy for å støtte testprosessen, inklusive å anbefale budsjettet for å velge verktøy (samt for mulig innkjøp og/eller vedlikehold), tildele tid og innsats for pilotprosjekter og gi kontinuerlig støtte i bruk av verktøy(ene)
- Beslutte oppsett av ett eller flere testmiljø
- Framsnakke og anbefale testerne, testteamene og fagfeltet testing innen organisasjonen
- Utvikle kompetansen og karrierene til testerne f.eks. gjennom opplæringsplaner, tilbakemeldinger om utført arbeid, ved veiledning osv.

Testlederrollen utøves på varierende måte avhengig av utviklingsmodellen. F.eks. vil det ved smidig utvikling være slik at noen av oppgavene listet over – særlig de daglige testoppgavene – blir håndtert innen teamet, ofte ved testeren i smidig-teamet. Noen av oppgavene som omfatter flere team, hele organisasjonen eller personalledelse, kan gjerne bli utført av testledere utenfor utviklingsteamene. Disse kalles av og til testrådgivere. Se Black 2009 for mer informasjon om styring av testprosessen.

Typiske oppgaver for testere:

- Granske og bidra til testplaner
- Analysere, granske og vurdere krav, brukerhistorier, akseptansekriterier, spesifikasjoner og modeller for testbarhet, dvs. testbasisen
- Identifisere og dokumentere testbetingelser og sette opp sporbarheten mellom testtilfeller, testbetingelser og testbasisen
- Designe, sette opp og verifisere testmiljø(ene) – ofte koordinert med systemadministrator og nettverksansvarlig
- Designe og skrive testtilfellene og testprosedyrene
- Forberede og framskaffe testdata
- Lage den detaljerte testkjøreplanen
- Utføre testene, evaluere resultatene og dokumentere avvik fra forventede resultater
- Bruke passende verktøy for å lette testprosessen
- Automatisere tester etter behov (kan bli støttet av en utvikler eller ekspert på testautomatisering)
- Vurdere ikke-funksjonelle kvalitetsegenskaper slik som ytelse, pålitelighet, brukbarhet, sikkerhet, kompatibilitet og portabilitet
- Granske tester laget av andre

Personer som jobber med testanalyse, testdesign, spesielle testtyper eller testautomatisering kan være spesialister i disse rollene.

Forskjellige personer kan ta over testerens rolle på ulike testnivåer, avhengig av utviklingsmodellen og risikobildet til produktet og prosjektet. F.eks. vil testerens rolle ofte utføres av utviklere under komponent- og komponentintegrasjonstesting. Under akseptansetest blir rollen til tester ofte utført av bedriftsanalytiker, fagekspert eller brukere. Under systemtest og systemintegrasjonstest blir testerens rolle ofte utført av et uavhengig testteam. Driftsakseptansetesting utføres ofte av driftspersonell og/eller systemadministratorer.

5.2 Testplanlegging og -estimering

5.2.1 Hensikt med og innholdet i en testplan

En testplan viser testaktivitetene for utviklings- og vedlikeholdsprosjekter. Planleggingen påvirkes av test-policyen og teststrategien til organisasjonen, utviklingsmodellen og metodene som brukes (se avsnitt 2.1), testomfanget, målene, risikoer, begrensninger, kritikalitet, testbarhet og de tilgjengelige ressursene.

Etter hvert som prosjektet og testplanleggingen går framover, blir mer informasjon tilgjengelig og flere detaljer kan legges til i testplanen. Testplanlegging er en kontinuerlig aktivitet og utføres i hele produktets livssyklus. (NB - Denne livssyklusen kan være lenger enn selve prosjektet, og også omfatte driftsfasen.) Tilbakemeldinger fra testaktiviteter bør brukes til å oppdage endringer i risikobildet slik at planene kan justeres. Planleggingen kan dokumenteres i en overordnet testplan og i tilhørende testplaner for testnivåer slik som systemtest og akseptansetest, eller for testtyper slik som testing av brukskvalitet og ytelsestesting.

Testplanleggingsaktiviteter kan inneholde følgende punkter, og noen av disse kan bli dokumentert i en testplan:

- Bestemme testomfanget, testmålene og testrisikoene
- Definere den overordnede testtilnærmingen
- Tilpasse og koordinere testaktivitetene med utviklingsaktivitetene
- Beslutte hva som skal testes, personer og andre ressurser som er påkrevd for å utføre ulike testaktiviteter, samt hvordan testaktivitetene skal utføres
- Lage tidslinjer for testanalyse, testdesign, testforberedelser, testkjøringer og vurderinger av testingen, enten med gitte datoer (f.eks. ved fossefallsmodell for utviklingen) eller for hver iterasjon (hhv. ved iterative utviklingsmodeller)
- Velge måleparametre for testovervåking og -styring
- Sette opp budsjett for testaktivitetene
- Bestemme detaljeringsgraden og strukturen på testdokumentasjonen, f.eks. ved å etablere maler og eksempler

Innholdet i testplaner varierer og kan inneholde mer enn temaene listet ovenfor. Eksempler på testplaner finnes i ISO-standarden ISO/IEC/IEEE 29119-3.

5.2.2 Teststrategi og testtilnærming

En teststrategi gir en generell beskrivelse av testprosessen, vanligvis på produkt- eller organisasjonsnivået. Vanlige typer teststrategier kan være:

- **Analytisk:** Denne typen teststrategi er basert på analyse av et moment, f.eks. krav eller risiko. Risikobasert testing er et eksempel på en analytisk tilnærming, der tester er designet og prioritert basert på risikonivået
- **Modellbasert:** I denne typen teststrategi er testene designet basert på en modell av et påkrevet aspekt ved produktet, slik som en funksjon, en forretningsprosess, en intern struktur eller en ikke-funksjonell kvalitetsegenskap, f.eks. pålitelighet. Eksempler på slike modeller er: forretningsprosessmodeller, tilstandsmodeller og statistiske modeller for å forutsi pålitelighet
- **Metodisk:** Denne typen teststrategi gjør systematisk bruk av forhåndsdefinerte samlinger av tester eller testbetingelser, slik som klassifisering av vanlige eller sannsynlige typer feil, en liste av viktige kvalitetsegenskaper eller bedriftsomfattende «se-og-føl»-standarder for mobile apper eller nettsider
- **I samsvar med prosessen eller standarden:** Denne typen teststrategi involverer å analysere, designe og implementere tester basert på eksterne regler og standarder, slik de er beskrevet i industri-spesifikke standarder, prosessdokumentasjonen, ved nitid identifikasjon og bruk av testgrunnlaget. Sist men ikke minst, ved å følge enhver prosess eller standard som er pålagt organisasjonen
- **Veiledet eller konsultasjonsbasert:** Denne typen teststrategi er primært drevet av rådgivere, veiledere eller interessentenes instruksjoner. Videre av eksperter på fagområdet eller teknologieksperter som kan være utenfor testteamet eller helt utenfor organisasjonen
- **Regresjonsmotvirkende:** Denne typen teststrategi er motivert av et ønske om å unngå regresjon, dvs. å sikre at feil ikke har blitt introdusert eller er aktivert i uendrete områder av eksisterende funksjonalitet. En slik teststrategi omfatter gjenbruk av eksisterende testmateriell (særdeles testtilfeller og testdata), omfattende automatisering av regresjonstester og standard testsuiter

- **Reaktiv:** I denne typen teststrategi er testingen reaktiv i forhold til komponenten eller systemet som blir testet, og hendelsene under testutføring heller enn å være forhåndsdefinert, slik som for de foregående teststrategiene. Tester blir designet og implementert underveis, og kan straks kjøres som en reaksjon på kunnskapen man har fått fra tidligere testresultater. Utforskende testing er en vanlig teknikk i reaktive teststrategier

En passende teststrategi blir ofte laget ved å kombinere flere av disse typene teststrategier. F.eks. kan risikobasert testing (en analytisk strategi) bli kombinert med utforskende testing (en reaktiv strategi). De utfyller hverandre og kan føre til mer effektiv testing når de brukes sammen.

Der teststrategien gir en generell beskrivelse av testprosessen, vil testtilnærmingen konkretisere den for et gitt prosjekt eller en leveranse. Testtilnærming er startpunktet for å velge testteknikker, testnivåer, testtyper og for å definere start- og sluttkriteriene (eller definisjonen av "ready" hhv. "done"). Hvordan teststrategien tilpasses er basert på beslutninger tatt ut fra kompleksiteten og målene til prosjektet, type produkt som utvikles og risikoanalysen for produktet. Den valgte testtilnærmingen er avhengig av konteksten og kan ta hensyn til risikoer, sikkerhet, tilgjengelige ressurser og kompetanse, teknologien, systemets natur, f.eks. skreddersydd versus hyllevare, testmålene og lovpålagte krav og forskrifter.

5.2.3 Start- og sluttkriterier

For å oppnå full kontroll på kvaliteten til programvaren og testingen på en effektiv måte, bør vi sette kriterier for når en gitt testaktivitet kan starte og når den er fullført. Disse kriteriene skal definere forutsetningene som må være oppfylt før det er hensiktsmessig å starte en testaktivitet. Startkriterier tilsvarer “definisjonen på klar til start” i smidig utviklingsmodell. Hvis vi begynner testingen uten at startkriteriene er oppfylt, vil vi sannsynligvis erfare at testaktiviteten er vanskeligere å gjennomføre, mer tidkrevende, mer kostbar og mer risikofylt. Sluttkriterier tilsvarer “definisjonen på fullført” i smidig utviklingsmodell. Sluttkriterier definerer hvilke betingelser som må være oppnådd for å erklære et testnivå eller en mengde tester for fullført. Start- og sluttkriterier bør defineres for hvert testnivå og hver testtype, og vil være ulike for ulike testmål.

Typiske startkriterier:

- Det finnes testbare krav, brukerhistorier og/eller testmodeller (dersom man følger en modell-basert teststrategi)
- Tilgjengelige testobjekter som har oppnådd sluttkriteriene for tidligere testnivåer
- Tilgjengelig testmiljø
- Tilgang til nødvendige testverktøy
- Tilstrekkelige testdata og andre nødvendige ressurser

Typiske sluttkriterier:

- De planlagte testene er kjørt
- Den planlagte testdekningen er oppnådd, f.eks. av krav, brukerhistorier, akseptansekriterier, risikoer og/eller koden
- Antall åpne feil er innenfor avtalte grenser
- Antall estimerte gjenværende feil er tilstrekkelig lavt
- Det er evaluert å være tilstrekkelig kvalitetsnivå av påliteligheten, ytelsen, brukbarheten, sikkerheten og andre relevante kvalitetsegenskaper

Selv uten at sluttkriteriene er møtt er det vanlig at testaktiviteter blir avsluttet pga. budsjettbegrensninger, tiden avsatt til testingen er ute og/eller tidspress for å slippe produktet ut i markedet. Det kan være akseptabelt å avslutte testingen under slike omstendigheter, gitt at prosjektets interessenter og produkteiere har gransket og akseptert risikoen forbundet med å gå i drift uten videre testing.

5.2.4 Kjøreplan

Når testtilfellene og testprosedyrene er ferdigstilt (noen testprosedyrer bør være automatisert) og samlet i testsuiter, så arrangeres disse i en kjøreplan som definerer i hvilken rekkefølge de skal utføres. Kjøreplanen bør ta hensyn til prioritet, avhengigheter, retester, regresjonstester og en mest mulig effektiv og rask utføring av testene.

Ideelt sett bør testtilfellene kjøres i rekkefølge etter prioritet, vanligvis ved å kjøre de med høyest prioritet først. Det er ikke alltid dette lar seg gjøre, fordi testtilfellene eller egenskapene har avhengigheter. Hvis et testtilfelle med høy prioritet er avhengig av et testtilfelle med lav prioritet, må testtilfellet med lav prioritet utføres først. Tilsvarende kan det være avhengigheter mellom testtilfeller og da må de ordnes i en hensiktsmessig (dvs. mest mulig effektiv) rekkefølge uavhengig av deres innbyrdes prioritet. Retest og regresjonstester må også prioriteres og inkluderes i kjøreplanen iht. hvor raske tilbakemeldinger på endringene som kreves. Også her kan det være avhengigheter.

I noen tilfeller er flere testrekkefølger mulig. I slike tilfeller må man gjøre avveininger mellom prioritet og testutføringens ressurs- og tidsbehov.

5.2.5 Faktorer som påvirker testinnsatsen

For å estimere testinnsatsen i et gitt prosjekt, en leveranse eller iterasjon, må man beregne arbeidsmengden som kreves for å nå målene for testingen. Faktorer som påvirker testinnsatsen: egenskaper ved produktet, utviklingsprosessen, kjennetegn ved personellet og testresultatene som vist under.

Produktegenskaper

- Risikoene ved produktet
- Kvaliteten av testbasisen
- Størrelsen på produktet
- Kompleksiteten til produktets fagområde
- Kravene til kvalitetsegenskaper, f.eks. sikkerhet og pålitelighet
- Påkrevd detaljeringsgrad for testdokumentasjonen
- Lovpålagte og andre forskrifts- og standardkrav

Kjennetegn ved utviklingsprosessen

- Organisasjonens stabilitet og modenhet
- Hvilken utviklingsmodell som brukes
- Testtilnærmingen
- Testverktøy
- Testprosessen
- Tidspress

Kjennetegn ved personellet

- Ferdigheter og erfaringer til de menneskene som er involvert, spesielt i forhold til liknende prosjekter og produkter, f.eks. kjennskap til fagområder
- Teamledelse og -samhold

Testresultater

- Antall feil og deres alvorlighetsgrad
- Mengden av påkrevde rettinger

5.2.6 Teknikker for testestimering

Det finnes flere estimeringsteknikker som brukes til å bestemme hvor mye innsats som trengs for god nok testing. De to vanligste estimeringsteknikkene er:

- Statistikkbasert teknikk: estimere testinnsatsen basert på målinger/statistikker fra tidligere, liknende prosjekter eller basert på typiske verdier
- Ekspertbasert teknikk: estimere testinnsatsen basert på erfaringen til eksperter eller de som eier testoppgavene

Ved smidig utvikling er bruk av «burndown»-diagram et eksempel på statistikkbasert tilnærming. Innsatsen måles og rapporteres og brukes deretter til å beregne teamets hastighet. Dette brukes til å estimere hvor mye arbeid teamet kan gjøre i neste iterasjon. På den andre siden er planleggingspoker et eksempel på ekspertbasert tilnærming der teammedlemmene estimerer innsatsen som kreves for å levere en produktegenskap basert på deres erfaring (ISTQB-AT Foundation Level Agile Tester Extension Syllabus).

For sekvensielle prosjekter er «defect removal»-modeller eksempler på statistikkbasert tilnærming, der antall feil og tiden det tar å fjerne dem måles og rapporteres. Dette danner så grunnlaget for å estimere framtidige prosjekter av tilsvarende natur. «Wideband Delphi»-estimeringsteknikk er et eksempel på en ekspertbasert tilnærming. I dette tilfellet gir ekspertgrupper estimater basert på deres erfaring (ISTQB-ATM Advanced Level Test Manager Syllabus).

5.3 Testovervåking og -styring

Hensikten med testovervåking er å samle informasjon for å gi tilbakemeldinger fra testingen og synliggjøre testaktivitetene. Informasjonen som skal overvåkes, kan samles inn manuelt eller automatisk. Den bør brukes til å vurdere testframdriften og måle om testens sluttkriterier eller testoppgavene iht. smidig-prosjekters definisjon av «done» er oppfylt. F.eks. at målene for testdekning av produktrisikoer, krav eller akseptansekriterier er nådd.

Teststyring omhandler enhver rettledning eller korrektive tiltak på grunnlag av måleresultatene og annen informasjon som er rapportert. Tiltakene kan omfatte enhver testaktivitet og kan påvirke enhver annen aktivitet i livssyklusen til programvaren.

Eksempler på teststyringsaktiviteter:

- Omprioritere tester når en identifisert risiko inntreffer, f.eks. forsinket leveranse av programvare
- Endre timeplanen for testingen pga. utilgjengelighet av testmiljø og andre ressurser
- Revurdere om en funksjon eller egenskap under test imøtekommer start- og/eller sluttkriterier etter retting

5.3.1 Målinger og statistikk brukt i testing

Målinger kan gjøres i løpet av og etter testaktiviteter for å evaluere:

- Framdrift i forhold til kjøreplanen og budsjettet
- Nåværende kvalitet til testobjektet
- Hvor godt testtilnærmingen passer
- Effektiviteten til testaktivitetene mht. testmålene

Vanlige testmålinger omfatter:

- Prosentandel utført arbeid i forhold til planlagt arbeid med å forberede testtilfeller (eller prosentandel av planlagte testtilfeller som er ferdig).
- Prosentandel planlagt arbeid utført for testmiljøforberedelser
- Utførte testtilfeller, f.eks. antall testtilfeller gjennomført/ikke-gjennomført, testtilfeller bestått/feilet og/eller testbetingelser bestått/feilet
- Informasjon om feil, f.eks. feiltetthet, feil funnet og rettet, feilraten og resultater av retest
- Testdekning i forhold til kravene, brukerhistorier, akseptansekriterier, risikoer eller koden/kodelinjer
- Fullførte oppgaver, ressursallokering og -bruk og innsats
- Kostnaden ved testingen, inklusive kostnaden sammenliknet med nytten av å finne neste feil eller kostnaden sammenliknet med nytten av å kjøre neste test

5.3.2 Hensikt, innhold og målgrupper for testrapporter

Hensikten med testrapportering er å oppsummere og kommunisere testaktiviteten både i løpet av og ved avslutningen av en testaktivitet, f.eks. et testnivå. Testrapporter lagd i løpet av en testaktivitet kalles gjerne framdriftsrapport for test, mens en testrapport lagd ved avslutningen av en testaktivitet er en sluttrapport for test.

Som en del av testovervåkingen og teststyringsaktivitetene lager testleder framdriftsrapport for test jevnlig for interessentene. I tillegg til innhold som er felles i framdriftsrapport og sluttrapport for test, kan framdriftsrapport for test typisk inneholde:

- Statusen til testaktivitetene og framdriften iht. testplanen
- Faktorer som hindrer testframdriften
- Testingen som er planlagt for neste rapporteringsperiode
- Kvaliteten til testobjektet

Når sluttkriteriene er oppnådd, produserer testleder sluttrapport for testen. Denne rapporten oppsummerer den utførte testingen basert på den nyeste framdriftsrapporten for testen og annen relevant informasjon.

Framdriftsrapport og sluttrapport for test kan inneholde:

- Oppsummering av den gjennomførte testingen
- Informasjon om hva som hendte i testperioden
- Avvik fra testplanen, inklusive avvik i kjøreplanen, varigheten eller innsatsen på testaktivitetene

- Teststatusen og produktkvaliteten mht. sluttkriteriene
- Faktorer som har blokkert eller fortsetter å blokkere framdriften
- Måledata om feil, testtilfeller, testdekning, testaktiviteters framdrift og ressursforbruk (slik som beskrevet i avsnitt 5.3.1)
- Restrisikoer (se avsnitt 5.5)
- Gjenbrukbare arbeidsprodukter fra testing

Innholdet i en testrapport vil variere avhengig av prosjektet, organisatoriske krav og livssyklusen. F.eks. vil et komplekst prosjekt med mange interessenter eller et strengt styrt prosjekt kreve mer detaljert og omfattende og strikt rapportering enn en rask programvareoppdatering. Et annet eksempel, fra smidig utvikling, er at framdriftsrapport for testingen kan innlemmes i oppgavetavlene, feilstatistikker og «burndown charts» og kan diskuteres i de daglige «stand-up» møtene, se ISTQB-AT Foundation Level Agile Tester Extension Syllabus.

I tillegg til å skreddersy testrapporter basert på prosjektets kontekst, bør de også tilpasses rapportens målgruppe. Typen og omfanget av informasjonen bør være forskjellig for en teknisk målgruppe eller et testteam, i forhold til det som vil bli tatt med i en testoppsummering til ledelsen. I det første tilfellet kan detaljert informasjon om typer feil og trender være viktig. I det andre tilfellet kan en overordnet rapport (f.eks. en statusoppsummering av feilene gruppert på prioritet, budsjett, tidsplan og testbetingelser fullført/feilet/ikke testet) være mer passende.

ISO-standarden (ISO/IEC/IEEE 29119-3) referer til to typer testrapporter: Framdriftsrapport for test og sluttrapport for test, og den inneholder strukturer og eksempler for hver type.

5.4 Konfigurasjonsstyring

Hensikten med konfigurasjonsstyring er å etablere og vedlikeholde integriteten til en komponent, et system, testmateriell og deres relasjoner til hverandre gjennom prosjektet og produktets livssyklus.

For å støtte testingen på en hensiktsmessig måte, kan konfigurasjonsstyring omfatte det å sikre følgende:

- Alle testelementer har en unik identifikator, er satt under versjonskontroll, endringer spores, og relasjoner er definert
- Alle elementer av testmateriellet har en unik identifikasjon, de er versjonsstyrt, endringer er sporet, og de har definerte relasjoner og er relatert til versjoner av testelementene slik at sporbarhet kan vedlikeholdes gjennom testprosessen
- Alle identifiserte dokumenter og programvareelementer er referert entydig i testdokumentasjonen

Som en del av testplanleggingen bør man identifisere og implementere prosedyrer for konfigurasjonsstyring og infrastrukturen (verktøy).

5.5 Risiko og testing

5.5.1 Definisjon av risiko

Risiko er muligheten for en framtidig hendelse som har negative konsekvenser. Risikonivået er bestemt av sannsynligheten for hendelsen og virkningen (skaden) fra den hendelsen.

5.5.2 Produkt- og prosjektrisiko

Produktrisiko

Produktrisiko omfatter muligheten for at et arbeidsprodukt (f.eks. en spesifikasjon, komponent, system eller test) kan mislykkes i å tilfredsstille legitime behov til dets brukere og/eller interessenter. Når produktrisiko er forbundet med spesifikke kvalitetsegenskaper til et produkt (f.eks. funksjonell egnethet, pålitelighet, ytelse, brukervennlighet, sikkerhet, kompatibilitet, vedlikeholdbarhet og portabilitet), kalles produktrisikoene også for kvalitetsrisikoer. Eksempler på produktrisikoer omfatter:

- Programvaren kan svikte i å utføre sine tiltenkte funksjoner i henhold til spesifikasjonen
- Programvaren kan svikte i å utføre sine tiltenkte funksjoner i henhold til brukerens, kundens og/eller interessentenes behov
- En systemarkitektur kan svikte i å støtte noen ikke-funksjonelle krav
- En bestemt beregning kan utføres feil under noen omstendigheter
- En løkke-struktur kan være programmert feil
- Svartider kan være utilstrekkelige for et system som krever høy ytelse
- Brukeropplevelsen kan avvike fra forventningene

Prosjektrisiko

Prosjektrisiko innebærer situasjoner som, hvis de oppstår, kan ha en negativ effekt på prosjektets evne til å nå sine mål. Eksempler på prosjektrisikoer omfatter:

- Prosjektproblemer:
  - Forsinkelser kan oppstå i leveranser, oppgavefullføring eller oppnåelse av sluttkriterier
  - Unøyaktige estimater, bruk av prosjektets midler til høyere prioriterte prosjekter eller generelt kostnadskutt i organisasjonen kan føre til utilstrekkelig finansiering
  - Endringer i siste liten kan føre til betydelige omarbeidelser
- Organisatoriske problemer:
  - Kompetanse, opplæring og bemanning kan være utilstrekkelig
  - Bemanningsutfordringer kan føre til konflikter og problemer
  - Brukere, forretningsressurser og fageksperter kan være utilgjengelige pga. prioriteringer i forretningsdriften som er i konflikt med prosjektbehov
- Politiske problemer:
  - Testere kan komme til å kommunisere sine behov og/eller testresultatene på en upassende måte
  - Utviklere og/eller testere kan la være å følge opp informasjonen funnet i testing og reviews, f.eks. unnlate å forbedre sin utviklings- og testpraksis
  - Det kan være upassende holdninger eller forventninger til testing, f.eks. ikke verdsette verdien av å finne feil i løpet av testing

- Tekniske problemer:
  - Krav kan være for dårlig definert
  - Kravene kan, gitt eksisterende begrensninger, ikke være oppfylt
  - Etablering av testmiljøet kan være forsinket
  - Datakonvertering, planlegging av migrering og verktøystøtte for dette kan være forsinket
  - Svakheter i utviklingsprosessen kan påvirke konsistensen og kvaliteten til arbeidsprodukter i prosjektet slik som design, kode, konfigurasjon, testdata og testtilfeller
  - Svak feilhåndtering og liknende problemer kan føre til opphopning av feil og annen teknisk gjeld
- Leverandørproblemer:
  - En tredjepart kan mislykkes i å levere et nødvendig produkt eller tjeneste eller gå konkurs
  - Kontraktsforhold kan forårsake problemer for prosjektet

Prosjektrisikoer kan påvirke både utviklings- og testaktiviteter. I noen tilfeller er prosjektledere ansvarlige for å håndtere all prosjektrisiko, men det er ikke uvanlig at testledere har ansvar for testrelaterte prosjektrisikoer.

5.5.3 Risikobasert testing og produktkvalitet

Risiko brukes for å fokusere den påkrevde testinnsatsen. Risiko brukes til å bestemme hvor og når testing skal starte og for å identifisere områder som trenger mer oppmerksomhet. Testing brukes til å redusere sannsynligheten for feil eller for å redusere effekten av feil. Testing er en risikoreduserende aktivitet som gir tilbakemeldinger om identifiserte risikoer og om gjenværende (ikke løste) risikoer.

Med risikobasert testing kan en proaktivt redusere produktrisikoer. Den er basert på analyse av produktrisikoer. Dette består av å identifisere produktrisikoer og vurdere sannsynlighet og konsekvens for hver risiko. Informasjonen om produktrisiko brukes deretter som veiledning for testplanlegging, testspesifikasjon, forberedelse og utførelse av testtilfeller, testovervåking og -styring. Det å analysere produktrisikoer tidlig bidrar til prosjektsuksess.

I en risikobasert testtilnærming brukes resultatene fra analysen av produktrisikoer til å:

- Bestemme hvilke testteknikker som skal brukes
- Bestemme spesielle testnivåer og hvilke testtyper som skal utføres, f.eks. sikkerhetstesting, tilgjengelighetstesting
- Bestemme hvor mye testing som skal utføres
- Prioritere testingen for å forsøke å finne de kritiske feilene så tidlig som mulig
- Bestemme om det kan benyttes andre aktiviteter i tillegg til testing for å redusere risiko, f.eks. gi opplæring til uerfarne designere

Risikobasert testing trekker veksler på den kollektive kunnskapen og innsikten til prosjektets interessenter gjennom å utføre analysen av produktrisikoer. For å sikre minimal sannsynlighet for at et produkt feiler, brukes risikostyringsaktiviteter som en disiplinert tilnærming til:

- Analyse (og jevnlige revurderinger) av hva som kan gå galt (risikoer)
- Bestemme hvilke risikoer som er viktige å ta hånd om

- Iverksette tiltak for å redusere disse risikoene
- Lage beredskapsplaner for å håndtere risikosituasjonene dersom de skulle oppstå

I tillegg kan testing komme til å identifisere nye risikoer, bidra til å bestemme hvilke risikoer som bør tas hånd om, samt redusere usikkerheten rundt risikoer.

5.6 Feilhåndtering

Siden en av hensiktene med testing er å finne feil, så bør feil funnet under testing loggføres. Hvordan feilene loggføres kan variere avhengig av konteksten til komponenten eller systemet som testes, testnivået og livssyklusmodellen. Enhver feil som blir funnet, bør undersøkes og spores fra oppdagelse og klassifikasjon til den er løst (f.eks. rettingen av feilene og vellykket retest av løsningen, utsettelse til en senere leveranse, aksept for dette som en varig produktbegrensing osv.) For å håndtere alle feilene fram til de er rettet, bør organisasjonen etablere en prosess for feilhåndtering med en arbeidsflyt og regler for klassifisering. Denne prosessen må det være enighet om blant alle som deltar i feilhåndteringen, inklusive designere, utviklere, testere og produkteiere. I noen organisasjoner kan loggføring og sporing av feil være svært uformell.

I løpet av prosessen for feilhåndtering kan noen rapporter vise seg å beskrive falske positive, dvs. misforståtte feilsituasjoner. F.eks. kan en test feile når en nettverksforbindelse er brutt eller utløpt. Denne situasjonen er ikke resultat av en feil i testobjektet, men er en uregelmessighet som må undersøkes. Testere bør forsøke å minimalisere antall falske positiver rapportert som feil.

Feil kan rapporteres under koding, statisk analyse, reviews, dynamisk testing eller ved bruk av et programvareprodukt. Feil kan rapporteres på grunn av problemer i koden eller de fungerende systemene, men også i enhver type dokumentasjon inklusive kravene, brukerhistoriene og akseptansekriteriene, i utviklingsdokumentasjon, testdokumentasjon, brukerveiledninger og installasjonsveiledninger. For å oppnå en effektiv og rask prosess for feilhåndtering, kan organisasjoner definere standarder for egenskaper, klassifikasjon og arbeidsflyten til feilene.

Typiske feilrapporter har følgende mål:

- Gi utviklere og andre parter informasjon om enhver feilsituasjon som oppstod for å sette dem i stand til å identifisere spesifikke virkninger, isolere problemet med en liten test som gjenskaper situasjonen, og korrigere de mulige feil(ene) etter behov – eller på annen måte løse problemet
- Gi testledere en mulighet for å følge opp produktkvaliteten og påvirkningen på testingen, f.eks. hvis det rapporteres mange feil, så vil testerne ha brukt mye tid på å rapportere dem istedenfor å kjøre tester og det vil være behov for mye mer retesting
- Gi ideer til forbedring av utviklings- og testprosessen

En feilrapport fra dynamisk testing inneholder vanligvis:

- Identifikasjon
- Tittel og en kort oppsummering av feilen
- Dato for feilrapporten, organisasjonen som utstedte den og forfatteren
- Identifikasjon av testelementet (det elementet i konfigurasjonen som ble testet) og testmiljøet
- I hvilke(n) fase(r) i utviklingslivssyklusen feilen ble oppdaget
- Beskrivelse av feilen for å kunne gjenskape og rette feilen, inklusive logger, situasjonsbilder av databasen, skjermbilder eller opptak (hvis slike er funnet under testkjøringen)

- Forventede og faktiske resultater
- omfanget til feilen og dens konsekvens (alvorlighetsgrad) for interessenten(e)
- Prioritet for retting
- Status til feilrapporten, f.eks. åpen, utsatt, duplikat, venter på å bli rettet, venter på å bli retestet, gjenåpnet, lukket
- Konklusjoner, anbefalinger og godkjenninger
- Sideeffekter utenfor testobjektet slik som andre områder som kan bli påvirket av en endring for å rette en feil
- Endringshistorikk slik som sekvensen av handlinger som prosjektmedarbeidere har tatt mht. feilen for å isolere, reparere og bekrefte den som rettet
- Referanser inklusive testtilfellet som avdekket problemet

Når man bruker verktøy for feilhåndteringen, kan noen av disse detaljene bli lagt inn automatisk slik som identifikator, oppgavetildeling, oppdatering av feilrapportens status i arbeidsflyten osv. Feil funnet gjennom statisk testing, spesielt reviews, vil normalt bli dokumentert på en annen måte, f.eks. i møtereferat fra reviews.

Et eksempel på innholdet i en feilrapport finnes i ISO standarden ISO/IEC/IEEE 29119-3 som refererer til feilrapporter som hendelsesrapporter «incident reports».

6 Verktøystøtte for testing — 40 minutter

Nøkkelord

datadrevet testing, nøkkelorddrevet testing, testautomatisering, testledelsesverktøy, verktøy for testutføring, ytelsestestverktøy

Læremål

6.1 Testverktøyvurderinger

FL-6.1.1 (K2) Klassifisere testverktøy i henhold til deres oppgaver og testaktivitetene som de støtter

FL-6.1.2 (K1) Identifisere fordeler og risikoer ved testautomatisering

FL-6.1.3 (K1) Huske spesielle vurderinger når det gjelder verktøy for testutføring og testledelse

6.2 Effektiv bruk av verktøy

FL-6.2.1 (K1) Identifisere hovedprinsippene for å velge et verktøy

FL-6.2.2 (K1) Gjenskape målene for å bruke pilotprosjekter til å introdusere verktøy

FL-6.2.3 (K1) Identifisere suksessfaktorene for evaluering, innføring, og installasjon av verktøy, samt support for testverktøyene, i en organisasjon

6.1 Generelt om testverktøy

Testverktøy kan bli brukt til å støtte en eller flere testaktiviteter. Slike verktøy inkluderer:

- Verktøy som er direkte brukt i testing, som f.eks. verktøy for testutføring og forberedelse av testdata
- Verktøy for å administrere krav, testscenarioer, testprosesser, automatiserte testscript, testresultater, testdata og feil, samt for rapportering og overvåking av selve testutførelsen
- Verktøy brukt til etterforskning/undersøking og evaluering
- Et hvert verktøy som støtter testing (regneark er også et testverktøy i denne sammenhengen)

6.1.1 Klassifisering av testverktøy

Testverktøy kan ha et eller flere av følgende formål avhengig av konteksten:

- Forbedre effektiviteten av testaktivitetene ved å automatisere gjentakende oppgaver eller oppgaver som er ressurskrevende hvis de utføres manuelt, f.eks. testutføring og regresjonstesting
- Forbedre effektiviteten av testaktivitetene ved å støtte manuelle testaktiviteter gjennom hele testprosessen (se avsnitt 1.4)
- Forbedre kvaliteten på testaktivitetene ved å støtte mer konsistent testing og bedre reproduserbarhet av feil
- Automatisere aktiviteter som ikke kan utføres manuelt, f.eks. storskala ytelsestester
- Øke påliteligheten av testing, f eks. ved å automatisere behandling av store datasett eller ved å simulere oppførsel

Verktøy kan bli klassifisert basert på flere kriterier som formål, prissetting, lisensmodell (f.eks. kommersiell eller åpen kildekode), og teknologi som er brukt. Verktøy er i dette pensum klassifisert etter testaktivitetene som de støtter.

Noen verktøy støtter tydelig én bestemt aktivitet, andre kan støtte flere aktiviteter. De blir klassifisert under den aktiviteten de er nærmest tilknyttet til. Verktøy fra en enkelt leverandør, spesielt de verktøyene som er designet til å fungere sammen, kan bli samlet i en felles verktøypakke/suite.

Noen typer verktøy kan være invaderende, ved at verktøyet selv har effekt på testresultatet. F.eks. kan responstiden for en applikasjon variere på grunn av ekstra instruksjoner som utføres i et ytelsestestverktøy. Nivået på testdekningen kan bli forvrengt ved å bruke et testdekningsverktøy. Konsekvensen av invaderende verktøy kalles verktøyeffekten eller instrumenteringseffekten.

Noen verktøy tilbyr støtte som er tilpasset utviklere (f.eks. ved komponent- og integrasjonstesting). Slike verktøy er merket med ”(D)” i avsnittene under.

Verktøystøtte for administrasjon av tester og testmateriell

Administrasjonsverktøy kan støtte enhver testaktivitet gjennom hele livssyklusen.

Eksempler på verktøy som støtter administrasjon av test og testmateriell:

- Verktøy for administrasjon av tester og applikasjonens livssyklus (engelsk: ALM)

- Verktøy for kravhåndtering, f.eks. sporbarhet til testobjekter
- Verktøy for feilhåndtering
- Verktøy for konfigurasjonsstyring
- Verktøy for kontinuerlig integrasjon (D)

Verktøystøtte for statisk testing

Verktøy for statisk testing er knyttet til aktivitetene og fordelene som er beskrevet i kapittel 3. Eksempler på slike verktøy:

- Verktøy som støtter reviews
- Verktøy for statisk analyse (D)

Verktøystøtte for testdesign og -implementering

Verktøy for testdesign hjelper til med å lage vedlikeholdbare arbeidsresultater i fasene testdesign og testimplementering, inklusive testtilfeller, testprosedyrer og testdata. Eksempler på slike verktøy:

- Verktøy for testdesign
- Verktøy for modell-basert testing
- Verktøy for å forberede testdata
- Verktøy for akseptansetestdrevet utvikling (eng: ATDD) og oppførselsdrevet utvikling (eng: BDD)
- Verktøy for testdrevet utvikling (eng: TDD) (D)

Noen ganger vil verktøy for å støtte testdesign og -implementering også kunne støtte testutføring og -logging. Deres resultat kan også sendes direkte videre til andre verktøy som støtter testutføring og -logging..

Verktøystøtte for testutføring og -logging

Mange verktøy skal støtte og forbedre aktiviteter for utføring og dokumentasjon av tester. Eksempler på slike verktøy:

- Verktøy for testutføring, f.eks. til å kjøre regresjonstester
- Verktøy for dekningsgrad, f.eks. dekning av krav og dekning av kode (D)
- Verktøy for å støtte testautomatisering (D)
- Verktøy for enhetstester (D)

Verktøystøtte for ytelsesmålinger og dynamiske analyser

Verktøy for ytelsesmålinger og dynamiske analyser er viktige for å støtte aktiviteter for ytelses- og lasttester siden disse aktivitetene ikke kan utføres manuelt på en effektiv måte. Eksempler på slike verktøy:

- Verktøy for ytelsestester

- Verktøy for overvåking
- Verktøy for dynamiske analyser (D)

Verktøystøtte for spesialiserte testbehov

I tillegg til verktøy som støtter den generelle testprosessen, er det mange andre verktøy for mer spesialisert testing. Eksempler på slike verktøy er verktøy som fokuserer på:

- Evaluering av datakvalitet
- Datakonvertering og -migrering
- Brukervennlighetstesting
- Testing av universell utforming
- Testing av lokale tilpasninger
- Sikkerhetstesting
- Portabilitetstesting, f.eks. teste programvare på tvers av flere støttede plattformer og systemer

6.1.2 Fordeler og risikoer ved testautomatisering

Å bare anskaffe et verktøy er ikke nok til å gi suksess. Hvert nytt verktøy som introduseres i organisasjonen vil kreve innsats for å oppnå reelle og varige fordeler. Det er fordeler og muligheter ved å bruke verktøy i testing, men det er også risikoer knyttet til det. Dette er spesielt tilfelle for testutføringsverktøy (som ofte omtales som testautomatisering).

Mulige fordeler ved å bruke verktøy for å støtte testutføring:

- Redusere gjentakende manuelt arbeid, f.eks. ved kjøring av regresjonstester, oppsett og nedtak av miljø, innsetting av samme testdata på nytt, og sjekk mot kodestandarder, noe som sparer tid
- Bedre konsistens/likhet og repeterbarhet/gjentakelse, f.eks. testdata opprettet på en ensartet måte, testene gjennomføres i den samme rekkefølgen med den samme frekvensen, og tester blir generert likt ut i fra definerte krav
- Flere objektive målinger, f.eks. statiske målinger, testdekning
- Enklere tilgang til informasjon om testingen, f.eks. statistikk og grafer for testenes progresjon, feilrate og ytelse

Mulige risikoer ved å bruke verktøy for å støtte testing:

- Forventningene til verktøyet kan være urealistiske, f.eks. i forhold til funksjonalitet og hvor enkelt det er å bruke
- Tid, kostnad og innsats for å introdusere verktøyet kan være underestimert, f.eks i forhold til opplæring og ekstern ekspertise og hjelp
- Nødvendig tid og innsats for å oppnå en merkbar og kontinuerlig fordel fra verktøyet kan være underestimert, f.eks i forhold til nødvendige endringer i testprosessen og kontinuerlig forbedringer i måten verktøyet brukes

- Innsatsen som kreves for å vedlikeholde dokumentasjon som genereres av verktøyet kan være underestimert
- Man kan stole for mye på verktøyet (kan bli sett på som en erstatning for testdesign eller testutføring, eller ved bruk av automatisert testing der hvor manuell testing ville vært bedre)
- Versjonskontroll av dokumentasjon kan bli oversett
- Forholdet og interoperabiliteten mellom ulike viktige verktøy kan bli oversett. Eksempler er verktøy for kravhåndtering, administrasjon av konfigurasjoner og feil, og verktøy fra ulike leverandører
- Leverandør eller produsent av verktøyet kan gå konkurs, legge ned verktøyet, eller selge verktøyet til en annen leverandør
- Leverandøren kan gi dårlig respons på forespørsler om hjelp (support), levere få oppgraderinger og feilrettinger
- Et open-source prosjekt kan bli utsatt
- En ny plattform eller teknologi er muligens ikke støttet av verktøyet
- Det er muligens ikke et klart eierskap for verktøyet, f.eks. for veiledning, oppdateringer, etc.

6.1.3 Spesielle vurderinger for verktøy til testutføring og -administrasjon

For å få en enkel og suksessfull introduksjon i en organisasjon, er det flere aspekter som må vurderes når man skal velge og integrere slike verktøy i en organisasjon.

Verktøy for testutføring

Verktøy for testutføring tester testobjekter (komponent eller system) ved å bruke automatiserte testscript. Denne typen verktøy krever ofte en betydelig innsats for å oppnå merkbare fordeler.

Å lage tester ved å ta opptak av manuelle tester virker attraktivt, men denne fremgangsmåten vil ikke skalere til et stort antall testscript. Et script basert på et opptak er en lineær representasjon med spesifikke data og hendelser som del av hvert steg. Denne typen script kan bli ustabilt når det skjer uventede hendelser. Siste generasjon av slike verktøy, som utnytter "smart" teknologi for bildegjenkjenning, har imidlertid økt brukbarheten for denne type verktøy. Samtidig vil de genererte scriptene være avhengig av vedlikehold så lenge systemets brukergrensesnitt utvikler seg over tid.

Datadrevet testing deler data inn i input til testingen og forventede resultater, vanligvis ved å bruke et regneark, og bruker et mer generelt testscript som kan lese inn inputdata og utføre det samme testscriptet med ulike data. Testere som ikke kan scriptspråket kan da lage nye testdata for disse pre-definerte scriptene.

Nøkkelorddrevet testing benytter nøkkelord for generelle scriptprosesser som beskriver hendelser/aksjoner som skal tas (også kalt aksjonsord). Disse aksjonene benytter da script som er basert på nøkkelordet til å behandle tilhørende testdata. Testere som ikke kan scriptspråket kan da definere tester ved å bruke nøkkelord og tilhørende data. Disse kan være skreddersydd til applikasjonen som testes. Flere detaljer og eksempler på datadrevet og nøkkelorddrevet testing er gitt i ISTQB-TAE Advanced Level Test Automation Engineer Syllabus, Fewster 1999 og Buwalda 2001.

Fremgangsmåtene nevnt over krever at noen kan scriptspråket som benyttes (testere, utviklere eller spesialister i testautomatisering). Uavhengig av teknikken scriptene bruker, så må de forventede resultatene for hver test bli sammenlignet med de faktiske resultatene fra testen - enten dynamisk (samtidig som testen kjører) eller lagret for senere sammenligning.

Verktøy for modellbasert testing (MBT) muliggjør at en funksjonell spesifikasjon kan bli omformet til en modell, som f.eks. et aktivitetsdiagram. Systemdesigneren gjør vanligvis denne oppgaven. MBT-verktøyet tolker modellen for å lage spesifikasjoner for testtilfeller som da kan bli lagret i et testledelsesverktøy og/eller utført av et verktøy for testutføring (se ISTQB-MBT Foundation Level Model-Based Testing Syllabus).

Verktøy for testledelse

Slike verktøy må ofte fungere sammen med andre verktøy eller regneark av ulike årsaker, f. eks.:

- Produsere nyttig informasjon på et format som oppfyller kravene i organisasjonen
- Bevare konsistent sporbarhet til krav i et verktøy for kravhåndtering
- Referere til testobjekters versjon i et verktøy for konfigurasjonsstyring

Dette er spesielt viktig å ta i betraktning når man bruker et integrert verktøy (f.eks. ALM) som inkluderer flere moduler som brukes av ulike grupper i organisasjonen. F.eks. verktøy benyttet til administrasjon av applikasjoners livssyklus, inklusive testledelse, feilhåndtering, prosjektstyring og budsjettinformasjon.

6.2 Effektiv bruk av verktøy

6.2.1 Hovedprinsipper for verktøyvalg

De viktigste vurderingene ved valg av verktøy for en organisasjon er:

- Organisasjonens modenhet - dens styrker og svakheter
- Identifisering av muligheter for en forbedret testprosess ved hjelp av verktøy
- Kjennskap til teknologiene som testobjektene bruker. Dette for å velge et verktøy som støtter teknologien
- Verktøy for bygging og kontinuerlig integrasjon som allerede er i bruk innen organisasjonen for å sikre kompatibilitet og integrasjon mellom verktøyene
- Evaluering av verktøyet mot klare krav og objektive kriterier
- Vurdering hvorvidt verktøyet er tilgjengelig for en gratis prøveperiode eller ikke (og for hvor lenge)
- Evaluering av leverandøren (inkludert opplæring, support og kommersielle aspekter) eller support for ikke-kommersielle (f.eks. åpen kildekode) verktøy
- Identifisering av interne krav til hjelp og støtte i bruken av verktøyet
- Evaluering av behov for opplæring med tanke på testkompetansen (og kompetanse på testautomatisering) til de som vil jobbe direkte med verktøyet
- Vurdering av fordeler og ulemper med ulike lisensmodeller, f.eks. kommersiell eller åpen kildekode
- Estimering av et kost-nytte forhold basert på et konkret «business case» (hvis nødvendig)

Som et siste steg burde man starte en «proof-of-concept» for å se hvorvidt verktøyet virker effektivt mot systemet som testes og nåværende infrastruktur. Det kan også være nødvendig å identifisere infrastrukturendringer som må gjøres for å kunne bruke verktøyet effektivt.

6.2.2 Pilotprosjekter for å introdusere et verktøy i en organisasjon

Etter å ha valgt verktøy og fullført en «proof-of-concept», vil en introduksjon av det valgte verktøyet i organisasjonen generelt starte med et pilotprosjekt som har følgende mål:

- Gi dybdekunnskap om verktøyet og forstå både dets styrker og svakheter
- Evaluere hvordan verktøyet passer sammen med eksisterende prosesser og praksiser, og bestemme hva som må endres
- Bestemme hvordan verktøyet skal brukes, administreres, lagres og vedlikeholdes. Det samme gjelder for testmateriellet, f.eks. bestemme navnestandarder for filer og tester, velge kodestandarder, opprette biblioteker og definere modulariteten av testsuitene
- Vurdere hvorvidt fordelene kan bli oppnådd til en overkommelig kostnad
- Forstå måledata som du ønsker at verktøyet skal samle inn og rapportere, og konfigurere verktøyet for å sikre at disse kan bli registrert og rapportert

6.2.3 Suksessfaktorer for verktøy

Suksessfaktorer for evaluering, implementering, utrulling, og support av verktøy innen en organisasjon inkluderer:

- Trinnvis utrulling av verktøyet til resten av organisasjonen
- Tilpasning og forbedring av prosesser for å passe med bruken av verktøyet
- Opplæring, rådgivning og veiledning for brukerne
- Definering av retningslinjer for bruken av verktøyet, f.eks. interne standarder for automatisering
- Implementering av informasjonsinnsamling om faktisk bruk
- Overvåking av bruken av verktøyet og dets fordeler
- Brukerstøtte for verktøyet
- Innsamling av erfaringer fra alle brukerne

Det er også viktig å sikre at verktøyet er teknisk og organisatorisk integrert i livssyklusen, noe som kan involvere andre organisasjoner som er ansvarlig for drift og/eller tredjeparts leverandører.

Se Graham 2012 for erfaringer og råd vedrørende bruk av verktøy for testutføring.

7 Referanser

Standarder

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

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

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

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

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

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

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

ISTQB dokumenter

ISTQB Glossary

ISTQB/NTB Terminologiliste på norsk (www.istqb-norge.no)

ISTQB Foundation Level Overview 2018

ISTQB-MBT Foundation Level Model-Based Tester Extension Syllabus

ISTQB-AT Foundation Level Agile Tester Extension Syllabus

ISTQB-ATA Advanced Level Test Analyst Syllabus

ISTQB-ATTA Advanced Level Technical Test Analyst Syllabus

ISTQB-ATM Advanced Level Test Manager Syllabus

ISTQB-SEC Advanced Level Security Tester Syllabus

ISTQB-TAE Advanced Level Test Automation Engineer Syllabus

ISTQB-ETM Expert Level Test Management Syllabus

ISTQB-EITP Expert Level Improving the Test Process Syllabus

Bøker 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*

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 referanser (ikke direkte referert i dette pensum)

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

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

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

8 Vedlegg A – Bakgrunn til pensum

Historien til dette dokumentet

Dette dokumentet er ISTQB Certified Tester Foundation Level Syllabus, grunnleggende nivå for internasjonal kvalifikasjon som er godkjent av ISTQB (www.istqb.org).

Dette dokumentet ble utarbeidet mellom 2014 og 2018 av en arbeidsgruppe bestående av medlemmer utnevnt av International Software Testing Qualifications Board (ISTQB). 2018-versjonen ble opprinnelig vurdert av representanter fra alle ISTQB nasjonale Boards, og deretter av internasjonale eksperter innen test.

Mål med kvalifikasjonen "Foundation Certificate"

- Få anerkjennelse for testing som et nødvendig og profesjonelt spesialfelt innen programvareutvikling
- Gi et standard rammeverk for utviklingen av en testers karriere
- Bidra til at profesjonelt kvalifiserte testere blir anerkjent av arbeidsgivere, kunder og arbeidskolleger, og øke testernes synlighet
- Sørge for utbredelse av sammenlignbar og god testpraksis innen alle områder av programvareutvikling
- Identifisere tema innen testing som er relevante og har en verdi i programvaremiljø
- Muliggjøre for firma som utvikler programvare å ansette sertifiserte testere og dermed å få kommersielle fordeler over deres konkurrenter ved at de annonserer deres ansettelseskrav for testere
- Gi en mulighet for testere og de som har interesse for testing å få en internasjonalt anerkjent kvalifikasjon i fagområdet

Mål med den internasjonale kvalifikasjonen

- Kunne sammenligne kunnskap om testing mellom forskjellige land
- Muliggjøre for testere enklere flytting mellom land
- Muliggjøre at multinasjonale/internasjonale prosjekter har en felles forståelse test
- Øke antallet kvalifiserte testere i verden
- Ha større gjennomslagskraft og verdi som et internasjonalt basert initiativ enn som et initiativ basert på et enkelt land
- Utvikle en felles internasjonal samling av forståelse og kunnskap om testing gjennom pensum og terminologilisten, og øke kunnskapsnivået om testing for alle deltakere
- Tilby testing som et profesjonelt yrke i flere land
- Muliggjøre for testere å få en anerkjent kvalifikasjon på sine nasjonale språk
- Muliggjøre deling av kunnskap og ressurser over landegrenser
- Sørge for internasjonal anerkjennelse av testere og denne kvalifiseringen gjennom deltakelse fra mange land

Startkrav for denne kvalifikasjonen

Startkriteriet for å ta eksamen for ISTQB Foundation Certificate in Software Testing er at kandidaten har en interesse for test av programvare. Men det anbefales på det sterkeste at kandidater også:

- Har i hvert fall en minimal bakgrunn enten i utvikling eller i testing av programvare, som f.eks. seks måneders erfaring som system- eller brukerakseptansetester eller som utvikler
- Tar et kurs som har blitt akkreditert i henhold til ISTQB's standarder (av et av de ISTQB-anerkjente nasjonale Boards)

Bakgrunn og historie for sertifikatet på grunnivå (Foundation Certificate in Software Testing)

Uavhengig sertifisering av testere av programvare startet i Storbritannia med British Computer Society Information Systems Examination Board (ISEB). Et "Software Testing Board" ble opprettet i 1998 (www.bcs.org.uk/iseb). I 2002 tok ASQF i Tyskland initiativ til en tysk sertifisering av programvaretestere (www.asqf.de). Pensum i dette dokumentet er basert på ISEB og ASQF sine. Innholdet er blitt reorganisert og oppdatert og noe nytt innhold er kommet til. Vekten er lagt på de tema som gir mest praktisk nytteverdi og hjelp for testere.

Et eksisterende grunnlagssertifikat (Foundation Certificate in Software Testing) fra f.eks. ISEB, ASQF eller et ISTQB-anerkjent nasjonalt Board, som er gitt før dette internasjonale sertifikat ble utgitt, blir ansett som ekvivalent til det internasjonale sertifikatet. Grunnlagssertifikatet (Foundation Certificate) utløper ikke og krever ikke fornyelse. Datoen for utstedelse av sertifikatet vises på sertifikatet.

I hvert deltakende land blir lokale forhold tatt hensyn til av et nasjonalt eller regionalt ISTQB-anerkjent Software Testing Board. Forpliktelsene disse nasjonale Boards skal oppfylle er spesifisert av ISTQB, men er implementert i hvert land. Forpliktelsene til et lands Board omfatter akkreditering av kursholdere og arrangering av eksamen.

9 Vedlegg B – Læremål / kunnskapsnivå

Følgende læremål er definert som tilhørende dette pensum. Hvert tema i pensum kan bli prøvd i eksamen tilsvarende sitt læremål.

Nivå 1: Huske (K1)

Kandidaten gjenkjenner, husker eller kan gjenta definisjonen av en term eller et konsept.

Stikkord: Huske, gjenkjenne, gjenta, observere, vite

Eksempel:

Kan gjenkjenne definisjonen av “failure” som:

- Feil eller fravikelse av en komponent eller et system fra dens forventede resultat, tjeneste eller output
- Hendelse der systemet eller komponenten ikke klarer å utføre sin spesifiserte funksjon eller å utføre den innenfor de begrensninger som er spesifisert

Nivå 2: Forstå (K2)

Kandidaten kan velge begrunnelsene eller erklæringene for påstander om dette tema og kan oppsummere, sammenligne, klassifisere, kategorisere og gi eksempler for dette testkonseptet.

Stikkord: oppsummere, generalisere, abstrahere, klassifisere, sammenligne, kartlegge, kontrastere, eksemplifisere, tolke, oversette, representere, trekke slutninger, konkludere, kategorisere, konstruere modeller

Eksempler

Kan forklare grunnen til hvorfor test analyse og design bør gjøres så tidlig som mulig:

- For å finne feil når de er billigere å ta bort
- For å finne de viktigste feilene først

Kan forklare likhetene og forskjellene mellom integrasjon- og systemtesting:

- Likheter: testobjekter for både integrasjon- og systemtesting omfatter flere komponenter og både integrasjonstesting og systemtesting kan omfatte ikke-funksjonelle testtyper
- Forskjeller: integrasjonstesting konsentrerer seg om grensesnitt og interaksjoner, mens systemtesting konsentrerer seg om aspektene som vedrører hele systemet, som prosessering fra start til slutt

Nivå 3: Bruke (K3)

Kandidaten kan velge den rette anvendelsen av et konsept eller en teknikk og anvende den for en gitt kontekst.

Stikkord: Implementere, utføre, bruke, følge en prosedyre, bruke en prosedyre

Eksempel

- Kan identifisere grenseverdier for gyldige og ugyldige partisjoner
- Kan velge testtilfeller for et gitt tilstandsdiagram for å dekke alle overganger

Referanser

(For de kognitive nivåer av læremålene)

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 Vedlegg C – Leveransebeskrivelse

ISTQB Foundation Syllabus 2018 er en stor oppdatering og omskrivning av 2011 versjonen. Derfor er det ingen detaljerte leveransenotater per kapittel eller avsnitt. Men i dette vedlegget blir de viktigste oppdateringene oppsummert. I tillegg er et separat Leveransenotat publisert som beskriver sporbarhet mellom læremålene i 2011-versjonen og læremålene i 2018-versjonen. Det kan man også se hvilke læremål er lagt til, oppdatert eller fjernet.

I begynnelsen av 2017 hadde mer enn 550 000 personer i mer enn 100 land tatt grunnivå-eksamen, og det er over 500 000 sertifiserte testere over hele verden. Gitt forventningen om at alle disse har lest grunnivå-pensumet for å kunne bestå eksamen, så er dette pensumet sannsynligvis til det mest leste dokumentet om programvaretest noensinne!

Oppdateringen er gjort med hensyn til denne arven og for å forbedre verdien ISTQB leverer til de neste 500.000 menneskene i det globale testsamfunnet.

I denne versjonen er alle læremål blitt redigert for å gjøre dem enkle og helthetlig.

Dermed oppnår vi en tydelig sporbarhet mellom hvert læremål og avsnittene som er relatert til det læremålet, samt eksamensspørsmålene. Denne sporbarheten går også motsatte veien: fra avsnittene og eksamensspørsmålene tilbake til læremålene.

I tillegg er minste undervisningstid for hvert kapittel gjort mer realistisk sammenlignet med tidsangivelsene i pensumutgaven av 2011 Tidsangivelsene er basert på analyse av læremålene som dekkes i hvert kapittel, og ved å bruke erfaringer og formler brukt i andre ISTQB-læreplaner.

Selv om dette er et grunnivå pensum, som uttrykker beste praksis og teknikker 'som har motstått tidens tann, har vi gjort endringer for å modernisere presentasjonen av materialet, spesielt når det gjelder programvareutviklingsmetoder, f.eks. Scrum og kontinuerlig leveranse og teknologier, f.eks. Tingens Internett.

Vi har oppdatert de refererte standardene for å gjøre dem mer aktuelle:
1. ISO / IEC / IEEE 29119 erstatter IEEE Standard 829.
2. ISO / IEC 25010 erstatter ISO 9126.
3. ISO / IEC 20246 erstatter IEEE 1028.

Siden ISTQB-porteføljen har vokst kraftig i løpet av det siste tiåret, har vi i tillegg lagt til omfattende kryssreferanser til relatert materiale i andre ISTQB-pensumer, der dette er relevant. Vi har også nøye gjennomgått samsvaret med alle pensumer og med ISTQB-ordlisten. Målet er å gjøre denne versjonen lettere å lese, forstå, lære og oversette, med fokus på økt praktisk brukbarhet og balansen mellom kunnskap og ferdigheter.

En detaljert analyse av endringene som er gjort i denne utgaven finnes i ISTQB Certified Tester Foundation Level Overview 2018 som kan lastes ned fra www.ISTQB.org.