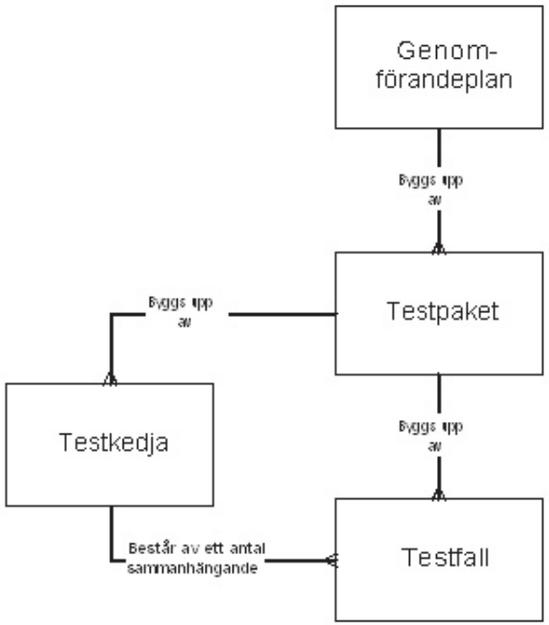
DEL III

Del III av boken tar upp praktiska detaljer om hur du organiserar dina testfall, vad du ska tänka på när du utför tester och förslag till hantering av avvikelser.

22 Organisera testfallen

Detta kapitel innehåller praktiska detaljer för hur du kan bygga upp och organisera testfall. Det finns en mängd olika sätt att göra detta på och jag väljer här att presentera ett upplägg som fungerat bra för mig i ett flertal projekt. Förutom att ta fram testfallen måste vi även knyta ihop dessa på lämpligt sätt så att resultatet blir körbart. Till vår hjälp har vi bland annat testkedjor, testpaket och genomförandeplan.

Det som gäller för alla testfall är att de har till uppgift att verifiera en eller flera aspekter av ett system. Utförandet börjar ofta med fristående enkla testfall för ett begränsat område som inte är beroende av varandra. Dessa kan köras parallellt utan inbördes ordning. En testkedja kan ses som en röd tråd som knyter ihop flera testfall som måste köras i en följd. Testpaket i sin tur är en praktisk uppdelning där du lägger alla testfall som av någon anledning behöver köras samtidigt i samma paket. Genomförandeplanen knyter ihop allting i ett detaljerat schema. Nästa bild förklarar hur det hänger ihop.



Figur 22.1: Bild över hur du kan organisera dina testfall. Det finns ingen absolut sanning hur det är bäst att organisera sina tester, som allt annat är det situationsberoende.

22.1 Testfall

Beskrivningen av ett testfall kan se olika ut beroende på vilken testnivå det ligger på. För testfall på en högre nivå kan det vara bra att använda en mall. Sättet som beskrivs här fungerar bra för system- och acceptanstest där testfallen innebär att användaren interagerar med systemet.

Det finns en spridd uppfattning om att ett testfall ska vara så detaljerat att vem som helst kan köra det. Tankesättet har sina brister. Det går inte att skriva ner allt som är viktigt och jag tror dessutom inte att vem som helst kan göra ett bra arbete med att utföra tester. I de flesta fall är det bättre att skriva mindre detaljerat av två viktiga anledningar. För det första tar det alldeles för mycket tid att skriva ner alla detaljer, för det andra sker det alltid förändringar under projektets gång så sannolikheten att de detaljer du skriver ner stämmer i slutändan är mycket små.

Det är viktigt att en verksamhetskunnig person deltar i testarbetet både vad det gäller design och utförande. Verksamhetspersonen är ofta inte testexpert vilket medför att det kan krävas mer detaljerad information i beskrivningen av testfallet.

Det finns absolut ingenting som säger att samtliga testfall ska vara dokumenterade på det här sättet. Ska du exempelvis testa en batchfunktion och varje testfall består av en rad i en fil så är det direkt olämpligt att tvinga in informationen i den mall som beskrivs nedan.

22.1.1 Storlek och uppdelning i testfall

Låt det praktiska styra. Maximal storlek bör vara ungefär så länge du klarar av att koncentrera dig eller får sitta ostörd i ett sträck. Skriv inte för stora testfall – då tar de för lång tid både att granska och att köra.

Om många aktörer är inblandade kan det vara lämpligt att dela upp testfallen efter vem som kommer att utföra dem.

Exempel: *En handläggare skapar ett ärende för kund som inte betalt månadsavgift på sitt bostadslån. Ärendet godkänns sedan av en kontorschef och beroende på skuldens storlek finns ytterligare en godkännandenivå hos en regionchef.*

I det har fallet är det mest praktiskt att ha tre testfall – en för varje inblandad aktör.

Du ska tänka på att testfallet ska ha ett tydligt syfte, och när syftet är uppnått är testfallet slut. Det är oftast praktiskt att ha flera olika verifieringar i varje testfall. Men har du alltför många steg i samma testfall riskerar du att det aldrig blir godkänt. Det finns ingen universallösning – använd erfarenhet, sunt förnuft och känsla.

22.1.2 Beståndsdelar

Administrativa fält

För att hantera testfallen finns ett antal administrativa fält. Det viktigaste är den unika identifieraren. Skriv en beteckning följt av en kort beskrivning för tydlighetens skull. Använder du en dokumentmall namnger du dokumentet med beteckningen. Tänk på namnsättning för framtida sortering. Vilka övriga fält du använder beror på vad som är mest praktiskt. Vanliga fält är: status, system, testnivå, prioritet och datum.

Exempel:

Id/Namn: *N10 Lägg upp en ny riskrapport för ett företag i åkeribranschen*

Status: *Klart för test*

Nivå: *Systemtest*

Syfte med testfallet

Här skriver du kortfattat ner syftet med testfallet.

Syfte*: Skapa ett ärende med ny risk på en existerande företagskund med god rating. Kunden betalar inte räntor och amorteringar på sina lån i två månader och hamnar då i vårt risksystem.*

Förutsättningar

Här definierar du vilka förutsättningar, förutom testdata, som gäller. Det kan gälla exempelvis andra testfall eller batchar som måste vara körda eller ett visst systemdatum.

Förutsättningar*: Systemdatum ska vara kvartal 1 2028 då detta testfall körs.*

Testdata

Här definierar du de data som måste finnas för att du ska kunna utföra testfallet. Det kan handla om data i systemet du just nu testar och om data i system som detta kommunicerar med. Du kan välja att beskriva testdata i ett dokument separat från testfallet. På så sätt kan du återanvända samma testfall flera gånger. Det är vanligt att först skapa testfallen och sen i efterhand fylla på med lämpligt testdata. Här bör du beakta vilka testdata du skriver för att det ska vara användbart vid regressionstester i framtiden. Exempelvis kan du istället för att skriva ett specifikt personnummer skriva ålder och kön, ett visst datum kan skrivas relativt dagens datum o s v.

Exempel:

Generellt*: Kunden måste vara ett företag och registrerat som åkeri och ha minst två registrerade avtal (lån). Testdata finns i separat dokument.*

Specifikt*: Använd kund 556778-4003 AB Sista Vilan.*

Teststeg

Beskriv stegvis de olika momenten i testfallet. Nivån på varje steg kan variera beroende på vilken typ av test som körs. Det är viktigt att tänka på att få det så praktiskt användbart som möjligt. Ett lämpligt tankesätt är att göra som i användningsfall, nämligen att ha ett steg för varje sak du utför och ett resultat som sen verifieras. Detaljeringsgraden passar bra för acceptanstest där ett begränsat antal testfall körs.

Resultat

Vissa saker kan du verifiera efter varje enskilt steg, andra efter att hela testfallet är kört. Det är viktigt att beskriva alla förväntade resultat innan du utför testerna – hur ska du annars veta om det är korrekt eller inte? Både utdata och systemets tillstånd ska beskrivas. Du kan verifiera ett testfall på många olika sätt, rapporter, fönster i applikationen och databasverktyg är några av sätten.

För det mesta är det bra att ta fram förväntat resultat då det finns en risk för feltolkningar om du tar fram svaren i samband med utförandet av testerna. Vid till exempel utforskande testning är det inte möjligt att ta fram svaren i förväg men detta ställer då höga krav på observationsförmågan hos den som utför testerna.

22.1.3 Exempel testfall för system- och acceptanstest

Id	N10 Lägg upp ny risk företag			
Syfte	Skapa ett ärende med ny risk på en existerande företagskund med god rating. Kunden betalar inte räntor och amorteringar på sina lån och hamnar då i vårt risksystem.			
System	DEKIS	Status	Godkänd för test	Författare	TORY01
Förutsättningar	Kunden måste vara ett företag. Systemdatum ska vara kvartal 1 2028 då detta testfall körs. Datum måste vara senast 11 mars.			
Testdata	Kund 556778-4003 AB Sista Vilan			
Steg	Beskrivning		Förväntat resultat	
1	Skapa nytt ärende		Du kommer till Nytt ärende dialogen	
2	Skriv in följande data: kundnr. Kund sedan 1997, Verksamhet: Begravningsbyrå, Orsak till problem: Ingen dör, Markera Konto 583376258 + Ändra			
3	Fyll i data kapital: 1796, Säkerhet Pi 1796/1706 100%, Tänkbar risk 1500, + För över till engagemang		Åter engagemangsbilden. Avtalet uppdaterat.	
4	Markera rapportorsak: Ny risk och periodrapport. Orsak: klarmarkering + OK		Ärendet sparas. Du är åter i listan över aktuella ärenden.	
Kontroll av resultat efter alla steg är körda				
1	Verifiera att ärendet visas korrekt i Listan över aktuella ärenden.		Status: klar för godkännande, rapportorsak: P,N, datum: idag	
2	Verifiera att ärendet visas korrekt i Visabilden: sök på status: klar för godkännande och rapportorsak: periodrapport		Status: klar för godkännande, rapportorsak: P,N, datum: idag	

Figur 22.2: Exempel på testfallsbeskrivning för system- och acceptanstest. På denna nivå kan det vara lämpligt att ha relativt detaljerade instruktioner för vad som ska fyllas i så att externa resurser kan hjälpa till med utförandet.

22.1.4 Fördelar och nackdelar med att dokumentera så här detaljerat

En av svårigheterna med test är att inget sätt passar bäst i alla lägen.

Fördelar med att beskriva varje detalj i testfallet är:

- Det kan sparas som regressionstestfall.
- Det kräver mindre kunskap om systemet som helhet.
- En utomstående person kan utföra det med lite stöd.
- Du får dokumentation av ditt arbete om detta behövs.
- Det går fortare att utföra testfallet då du bestämt detaljerna i förväg.

Nackdelar med att ha alltför detaljerade testfall kan vara att:

- När du väl ska köra testfallet upptäcker du att systemet har förändrats och detaljerna inte längre stämmer.
- Efter att ha kört ett antal testfall har du så pass mycket kunskap att du inte längre läser allt som står beskrivet och riskerar då att göra fel genom att du missar vissa detaljer.
- Det tar en massa extra tid att skriva ner detaljerna, tid du kunde ha använt till att designa fler testfall eller utföra tester.

22.2 Testkedjor

Efterhand som de enkla funktionerna är verifierade måste du även verifiera att hela applikationen, även manuella rutiner, hänger ihop. I det fallet blir det ofta alltför komplicerat och opraktiskt att skriva allt i ett enda testfall. Du kan då knyta ihop flera testfall i kedjor. En testkedja består alltså av ett antal testfall som körs i en viss ordningsföljd. Det är som en röd tråd som löper genom systemet under en viss tidsperiod. Det är lämpligt att dela upp testfallet i en kedja om flera olika personer ska utföra olika delar, om vi måste ändra datum i systemet mellan olika steg eller om helt olika dialo-

ger/funktioner i systemet används för olika steg. Testkedjor passar speciellt bra för att verifiera verksamhetsprocesser, större användningsfall, scenarier samt vid data- och affärscykeltest.

22.2.1 Beståndsdelar

Administrativa fält

Även för testkedjor är det bra att ha några administrativa fält. En unik identifierare är det viktigaste.

Exempel: N1 Ny risk företag

Syfte med testkedjan

Här skriver du ner den översiktliga beskrivningen av syftet med testkedjan.

***Syfte**: Övergripande syfte är att studera ett ärendes livscykel från att det skapas till dess att det avslutas. En existerande företagskund med god rating betalar inte räntor och amorteringar på sina lån. Då skapas ett ärende med ny risk och samtidigt en periodrapport. Ärendet följs upp kvartalsvis med ökande och minskade risker och avslutas med att kunden betalar tillbaka och byter bank – avslutar alla konton.*

Testdata

Här definierar du de data som måste finnas för att du ska kunna utföra testfallet. Det kan handla om data i systemet du just nu testar och om data i system som detta kommunicerar med. Ofta har du tagit fram specifikt testdata för just detta testfall och då specificeras detta här.

***Testdata**: Kunden måste vara ett företag. Använd kund 556778-4003 AB Sista Vilan genomgående i hela testfallet. Kunden rapporteras av kontor 6983-2 som ligger i region 8.*

Förutsättningar

Här definierar du vilka förutsättningar, förutom testdata, som gäller. Det kan gälla exempelvis andra testfall eller batchar som måste vara körda eller ett visst systemdatum.

***Förutsättningar**: Testerna börjar kvartal 1 2028 och avslutas i kvartal 1 2029.*
Det måste finnas handläggare och kontorsansvarig på kontor 16983-8.
Det måste finnas en regionansvarig på region 8.
Handläggare på central nivå är ej med i detta scenario.

Testfall

Här gör du en lista över de testfall som ingår i testkedjan och i vilken ordning de körs.

***Hela kedjan**:*
Q1 2028
Testfall: N10 Ny risk företag
Testfall: N11 Godkänn periodrapport Q1 2028
Q2 2028
Testfall: N12 Risk förvärras till sannolik förlust
Testfall: N13 Godkänn periodrapport Q2 2028
Q3 2028
Testfall: N14 Sannolik förlust vid återbetalning
Testfall: N13 Godkänn periodrapport Q3 2028
Q4 2028
Testfall: N15 Ärendet avslutas då kund betalar alla skulder och byter bank
Testfall: N16 Godkänn periodrapport Q4 2028
Q1 2029
Testfall: N17 Verifiera ärendets status i systemet efter årsskiftet

22.2.2 Resultat

Om du vill göra specifika kontroller av hela testkedjan fyller du i kontrollpunkterna här. Många av verifieringarna görs dock i samband med varje enskilt testfall.

22.2.3 Hur stor är en testkedja?

En testkedja kan ses som en röd tråd som löper genom systemet. Hur lång tråden är och vad den innehåller varierar beroende på system. Några typiska exempel på hur du kan få fram testkedjor är tidscykler och verksamhetsprocesser.

22.3 Testpaket

Ett paket innehåller testfall som det är praktiskt eller nödvändigt att köra vid samma tidpunkt. När du sen skriver din genomförandeplan skriver du ofta ner vilka testpaket som körs varje dag. Ofta är det inte praktiskt möjligt att köra alla testfall i en kedja på samma gång – det kan krävas batchar, datumrullningar eller annat mellan olika delar.

22.3.1 Beståndsdelar

Id

Namnge testpaketet med en unik identifierare på ett sätt som är lätt att förstå.

Paket:

Testkedjor Q1 2028 (innehåller samtliga testfall som körs under kvartal 1. Det är praktiskt då rapporteringen sker kvartalsvis. För varje kvartal sker godkännande och speciella batchprogram körs.

Innehåll

Testpaket är ett sätt att gruppera testfall du måste eller bör köra vid ett visst tillfälle, samtidigt eller i en viss följd. Det finns en klar fördel att lägga upp alla testfall som ska köras samma dag i samma testpaket – det blir då lättare att administrera och utföra testerna. Testpaketen kan till exempel vara uppdelade efter datum. Du får själv avgöra vad som passar i just ditt projekt.
För att verifiera verksamhetsprocesserna i projektet skapade vi fem testpaket. Varje testpaket innehöll alla de testfall som körs ett visst kvartal för alla verksamhetsprocesser.

Testkedjor Q1 2028
Testkedjor Q2 2028
Testkedjor Q3 2028
Testkedjor Q4 2028
Testkedjor Q1 2029

22.3.2 Hur vet du vad som är ett bra testpaket?

Återigen handlar det om vad som är praktiskt i ditt fall. I exemplet ovan paketerade vi efter datum. Detta är ofta, men inte alltid, en bra indelning. I tidiga nivåer av ett projekt kan du välja att paketera efter testområde (funktion, dialog, objekt etc). Ett paket kan då innehålla exempelvis en genomgång av samliga hjälptexter i systemet eller alla detaljtester av ett speciellt fönster.

22.4 Genomförandeplan

För att i detalj planera testernas utförande kan du för varje testnivå skapa en genomförandeplan. Denna innehåller information om vem som ska göra vad och vid vilken tidpunkt.

22.4.1 Innehåll

Förberedelse

Vanligen börjar nivån med förberedelser, t ex ladda grunddata, ställ in parametrar, verifiera testmiljön. När du har lyckats genomföra förberedelserna korrekt är det att rekommendera att ta en backup av läget. Kommande tester börjar sen från backupen och du slipper göra om förberedelserna flera gånger. Om det sker förändringar som påverkar de förberedande aktiviteterna måste du självklart undersöka om det är nödvändigt att regressionstesta förberedelserna.

Happy day

För att kontrollera om applikationen är redo att testas kör du ofta några inledande enkla tester. *Happy Day*-tester som de ibland kallas är en inledande kontroll av att systemet hänger ihop. Använd de centrala funktionerna med enkel data. Här ligger ofta en beslutspunkt som säger att du inte går vidare med ytterligare tester innan de mest grundläggande är godkända. Dessa inledande tester börjar du lämpligen med vid varje ny större leverans.

Som kuriosa kan nämnas att detta även kallas *Smoke-test* vilket stammar från den gamla goda tiden då man kontrollerade kretskort som vid felkonstruktion kunde börja brinna med rökutveckling som resultat.

Nästa steg

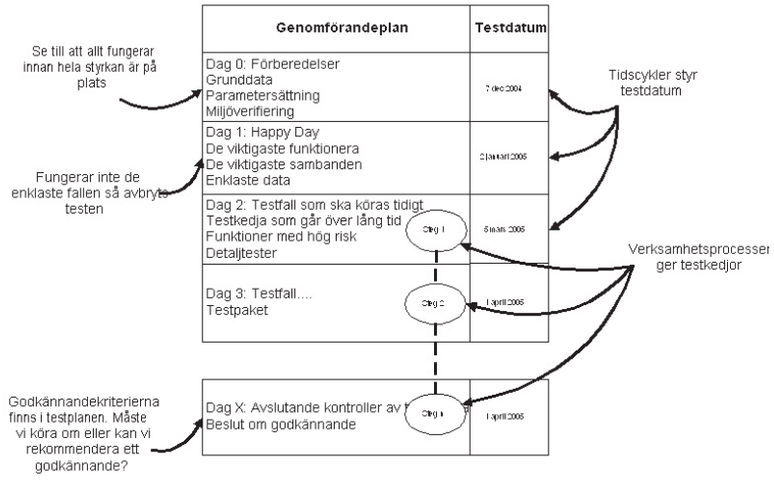
Kommande tester kan delas upp på många olika sätt. För tidiga nivåer är det vanligt att först verifiera enskilda dialoger, funktioner eller testområden i detalj innan du går vidare med mer övergripande scenariobaserade tester. Man bygger upp schemat efter vad som är mest praktiskt. Parametrar som påverkar:

- Prioriteten på testområden – det viktigaste testas först.
- Den ordning som leveranserna kommer i – kan skilja från testernas prioritet.

- Flödet genom systemet kan kräva att vissa funktioner utförs innan andra – verksamhetsprocesserna är en viktig källa till information.
- Tidscykler styr vilka testdatum som gäller.

Ett vanligt synsätt är att först se till att grundläggande tester görs för hela systemet innan du börjar med testkedjor för de viktigaste delarna.

22.4.2 Exempel genomförandeplan översikt



Figur 22.3: Genomförandeplan översikt för testnivå. Det underlättar att ha någon form av översiktlig plan, speciellt om vi är beroende av andra system och resurser.

22.4.3 Exempel genomförandeplan i detalj

Dag 1			
Aktivitet/testpaket	Tid	Ansvarig	Kommentar
Verifiera testmiljön	10-Aug	Leena	Checklista testkedjor.xls
Testkedjor Q1 2028	17-Oct	Torbjörn, Fredrik	
Beslutspunkt	17.00	Leena	Klara att gå vidare till Q2?
Dag 2			
Testkedjor Q2 2028	17-Sep	Torbjörn, Fredrik	
Beslutspunkt	17.00	Leena	Klara att gå vidare till Q3?
Dag 3			
Testkedjor Q3 2028	17-Sep	Torbjörn, Fredrik	
Beslutspunkt	17.00	Leena	Klara att gå vidare till Q4?
Dag 4			
Testkedjor Q4 2028	17-Sep	Torbjörn, Fredrik	
Beslutspunkt	17.00	Leena	Klara att gå vidare till Q1 2029?
Dag 5			
Testkedjor Q1 2028	15-Sep	Torbjörn, Fredrik	
Beslutspunkt	15.00	Leena	Godkänna testkedjan eller omstart Q1?

Figur 22.4: Genomförandeplan på detaljnivå. Som testledare vill jag gärna kunna planera in vem som ska göra vad och när.

23 Avvikelsehantering

Syftet med att skriva en avvikelserapport är att få den åtgärdad Cem Kaner

Den bästa testaren är inte den som skriver flest avvikelserapporter utan den som får de flesta rapporterna åtgärdade. Med avvikelse avses alla fall där resultatet av en test skiljer sig från det förväntade. Orsaken till avvikelsen behöver inte alltid vara programfel, andra orsaker är tolkningen av kravdokument eller problem med testfallen. Defekt, bugg, felrapport eller incident är andra benämningar som används.

Det främsta syftet med att skapa en avvikelserapport är att få de fel du hittar rättade. Detta gör avvikelserapporterna till det absolut viktigaste du skapar under testexekveringen. Avvikelsehanteringen är en central process och det är viktigt att den fungerar effektivt. [100] Verktygsstöd och process kan även användas till allmän ärendehantering i ett projekt och med fördel tas databasen med testfall och avvikelser över vid överlämnande till förvaltning. Det finns mycket att vinna på att bygga vidare på samma grund vid nya releaser av systemet.

Det finns flera skäl till varför det är intressant att ha en standardiserad avvikelsehanteringsprocess: [101]

[100] Black, Rex [1999]: Managing the Testing Process
[101] Black, Rex [1999]: Managing the Testing Process, s 107-111

- Kontroll på alla avvikelser.
- Gemensamt arbetssätt ger snabbare hantering.
- Bidra till att ge en tydlig bild av nuläge och kvalitet i en version av en produkt.
- Underlätta uppföljningen av projekt.
- Underlag till förbättringsarbete.
- Medge jämförelser mellan olika projekt.
- Skapa gemensamma rutiner för statistik och rapportering.

23.1 Verktygsstöd

Om det finns något läge där jag verkligen rekommenderar ett verktyg är det vid hantering av avvikelser. Tiden då vi hanterade avvikelser via ostrukturerade mejl eller som dokument i en folder borde vara över nu. Det finns en rik flora av verktyg, både kommersiella och gratis. Oavsett vad ni väljer att satsa på – se till att det finns något verktyg för att underlätta arbetet. En liten organisation behöver inte köpa det dyraste på marknaden utan nöjer sig med en billigare version som fyller sitt syfte.

Fördelarna med att använda ett verktyg med en central hantering av data är bland annat att:

- Det sparar tid.
- Kommunikationen mellan parterna blir bättre och oavsett var du sitter så finns det en gemensam hantering av avvikelser och du ser alltid rätt version.
- Analys och rapportmöjligheter gör det enkelt att snabbt ta fram en nulägesbeskrivning och en prognos.
- Central prioritering ger fokus på rätt avvikelser.
- Det är enklare att administrera aktuella ärenden.
- Historik i databasen ger bättre analys och uppföljningsmöjligheter.

23.2 Arbetsprocess

Det är viktigt att anpassa arbetsprocessen så att den passar er. Ibland kan det finnas behov av en mycket formell detaljerad process om många personer är inblandade, i andra fall ses en detaljerad process som ett hinder. Kortfattat kan man dela in livscykeln för en avvikelse i fyra olika steg.[102]

23.2.1 Avvikelsen identifieras

Skapa en rapport och dokumentera stegen för att återskapa situationen. Vad är fel och hur allvarligt är det? All information som underlättar vidare arbete är bra – allt annat är dåligt. Notera om felen uppträder oregelbundet och inte går att återskapa.

23.2.2 Analys och beslut om åtgärd

Analys och prioritering sker ofta av en grupp sammansatt av projektledare, testledare, utvecklingsledare och ofta även beställaren. Prioriteten är det styrmedel som projektet har för att bestämma vad som ska rättas först. Håll prioriteringslistan över akuta åtgärder kort – annars blir den ignorerad. Här gäller det att ha en smidig process för att inte stoppa upp arbetet. Allvarliga fel kan snabbt slussas ut till ansvarig person utan att hela prioriteringsgruppen måste sammanträda. Fokus bör vara på de saker som är oklara. Hur formell gången är kan variera under projektets gång. Ta beslut om rapporten ska åtgärdas, avvisas utan åtgärd eller tas om hand vid ett senare tillfälle. Utse ansvarig person.

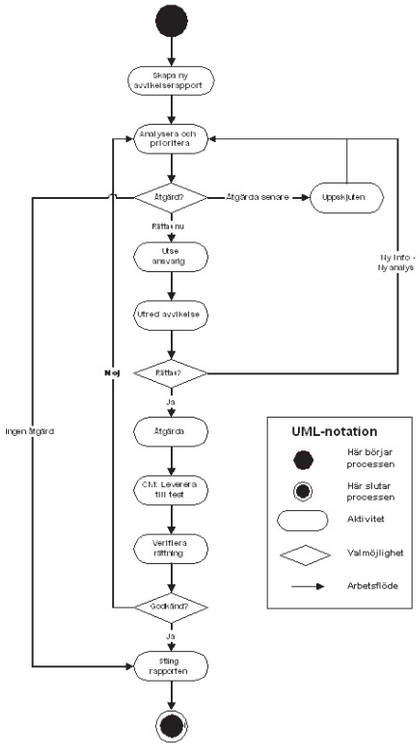
23.2.3 Utredning och åtgärd

Ansvarig person utreder avvikelsen och tar reda på vad som är orsaken. Ansvaret kvarstår till dess åtgärd är utförd eller avvikelsen delegeras till annan person. I mindre projekt är det mer praktiskt att diskutera med övriga projektmedlemmar än att ställa en fråga via rapporten och ändra ansvarig person. I större projekt kan det vara tvärtom.

[102] IEEE 1044 Standard for Incidents

23.2.4 Verifiera rättningen

Innan verifieringen sker är det viktigt att kontrollera att rättningen inte bara är utförd utan att den nya koden är lyft till testmiljön. En bra process för konfigurationshantering hindrar att du testar på fel kod. Testaren som hittade felet eller testledaren stänger rapporten då den är verifierad. Fundera på vilka andra testfall du bör köra för att verifiera delar som kan ha påverkats av rättningen.



Figur 23.1: Förslag till arbetsprocess för hantering av avvikelser. Anpassa processen till era lokala förhållanden. Gör den så pass enkel att alla följer den utan stora protester, men tillräckligt detaljerad för att få bra kontroll.

Anpassa processen till era lokala förhållanden. Gör den så pass enkel att alla följer den utan stora protester men tillräckligt detaljerad för att få bra kontroll.

23.3 Innehåll i avvikelserapport

Viktiga detaljer att ha med är följande:

23.3.1 Rubrik

Skriv något kortfattat och relevant så att du och alla andra inblandade personer lätt kan identifiera rapporten.

23.3.2 Upptäckt datum/person

Viktigt att veta vem som skapat rapporten och när detta är gjort.

23.3.3 Ansvarig person

Det ska alltid finnas någon som är ansvarig för nästa steg i hanteringen.

23.3.4 Klassning

Hur allvarligt är felet? Ha inte för många graderingar, tre till fem olika nivåer brukar vara lämpligt. Ett förslag är följande:

1. **Stoppande** – förlust av data, säkerhetsproblem, krasch – hela eller stora delar av systemet är obrukbart om detta fel finns kvar. Under test: Testerna kan ej fortsätta. Under produktion: Produktionsstoppande.
2. **Allvarlig** – Funktionsstoppande, större funktionsfel utan möjlighet till alternativlösning.
3. **Mindre allvarligt** – ett mindre funktionsfel eller ett allvarligt fel där det finns möjlighet till alternativlösning.
4. **Skönhetsfel** – påverkar ej funktionaliteten – felstavning, färg och form.
5. **Ogiltig** – stängd utan åtgärd, ska ej räknas med i statistiken (framkommer efter analys).

23.3.5 Beskrivning av avvikelse

Skriv ner stegen som ska utföras för att återskapa felet – det är mycket vik-

tigt att vara väldigt tydlig här. Om inte du själv kan återskapa felet hur kan då den som ska analysera avvikelsen göra det?

Isolering av felet – om det är möjligt så underlättar det mycket vid en senare analys om du kan beskriva den specifika situation där något går fel. Det kan handla om specifikt testdata, en viss font, en kombination av händelser, första gången du gör något etc.

Lägg gärna in en bilaga med en skärmkopia om detta underlättar förståelsen.

23.3.6 Beskrivning av åtgärd

Detta fält fylls i under utredning/rättning/test och beskriver vilka ändringar som gjorts för att åtgärda felet.

23.3.7 Prioritet

Ett kompletterande fält för att styra arbetet. Sätts ofta till *Hög*, *Normal* eller *Låg*. Skillnaden mot klassning är att en avvikelse kan tillhöra ett för tillfället lågt prioriterat område även om det är allvarligt och tvärtom. Klassning är en objektiv värdering av ett fel och prioritering ett sätt att välja vad som ska rättas först.

23.3.8 Typ av fel

För att vi ska ha möjlighet att lära oss av våra misstag är det nödvändigt att analysera vad det var som gick fel. Detta gör vi genom en analys av orsakerna till varje rapport. En avvikelse kan bero på så vitt skilda saker som dokumentation, krav, kodning, design eller test.

23.3.9 Övriga fält

Andra intressanta fält som kan användas är om felet är reproducerbart eller ej, vilken del av systemet det ligger i, testnivå, miljö och version av programkoden. För att få en smidig hantering är det viktigt att inte har för många fält att fylla i, risken är då att det blir för tungt att hantera. Det är oftast bäst med en enkel process och förhållandevis få fält.

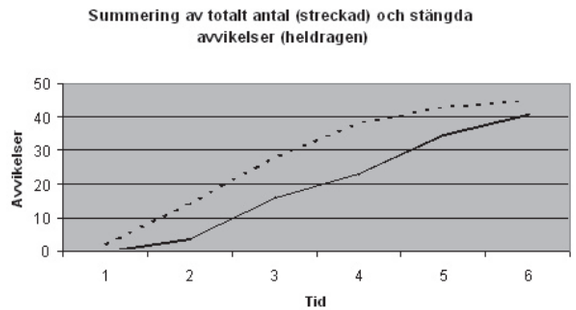
23.4 Uppföljning

Under tiden testningen pågår kan vi följa upp arbetet genom att mäta mot testfall eller avvikelserapportering. Information används för att ge projektet en bild över aktuell status och ligger som underlag till vårt beslut att avsluta testerna.

För att få en grundläggande statusrapport över test föreslår jag två mycket enkla diagram att börja med.

I. Antalet avvikelser per vecka, testomgång eller testnivå.
II. Antalet testfall totalt, körda, godkända eller misslyckade.

23.4.1 Avvikelser

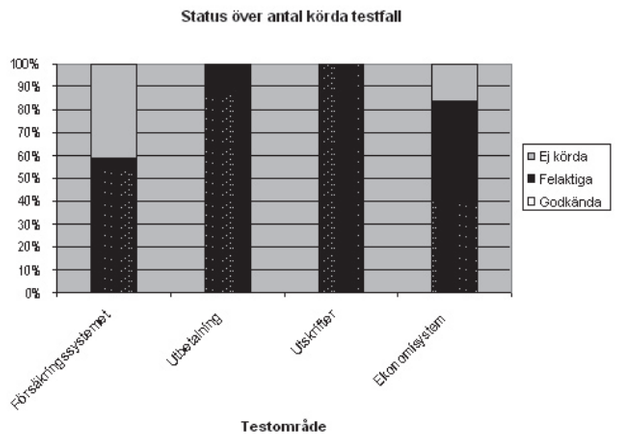


Figur 23.2: Summering av avvikelser över tiden. Den klassiska analysen av diagrammet är att testerna börjar närma sig sitt slut då det totala antalet avvikelser ska sluta växa. Detta innebär att den övre kurvan planar ut. Dessutom ska den undre kurvan som visar antalet stängda avvikelser närma sig den övre kurvan vilket betyder att de fel som hittats är rättade och verifierade.

Den klassiska analysen av diagrammet är att testerna börjar närma sig sitt slut då det totala antalet avvikelser aslutar att växa. Detta innebär att

den övre kurvan planar ut. Dessutom ska den undre kurvan som visar antalet stängda avvikelser närma sig den övre kurvan vilket betyder att de fel som hittats är rättade och verifierade. En mer detaljerad bild får du genom att ta hänsyn till klassningen på avvikelserna. Speciellt viktigt är det att noga analysera de allvarligaste felen.

23.4.2 Antal testfall



Figur 23.3: Summering av testfall och status per område. Denna graf visar hur projektet mår, vilka områden som ligger efter och vilka som går bättre.

Även här finns ett antal saker att analysera. Antalet utförda testfall totalt jämfört med det planerade antalet visar hur långt vi har kvar av utförandet. Både antalet felaktiga testfall i sig själv och i relation mot antalet godkända testfall visar hur bra kvalitet det är på det som levererats. En mer detaljerad analys tar hänsyn till vilken prioritet testfallen har och hur komplicerade de är.

23.4.3 Fallgropar

Det finns en tjusning med enkelheten i diagrammen, samtidigt finns det några uppenbara fallgropar. Om vi bara visar ett diagram över antalet avvikelser utan relation till antalet testfall så vet vi inte orsaken till att det ser ut som det gör. Att vi hittar färre avvikelser eller lyckas med fler testfall kan bero på att vi kör färre testfall, att vi kör samma testfall som förut för att verifiera rättningar eller att det område vi testar just nu är enklare än de andra och att vi därför hittar procentuellt färre fel. Det kan också bero på att vi inte fått nya leveranser av kod eller att någon varit sjuk och att vi därför inte fortsatt att testa i samma omfattning. Statistik kan ge värdefull information men är inte den slutliga sanningen. Felaktig statistik kan dessutom vara direkt skadlig om den ger en missvisande bild av läget.

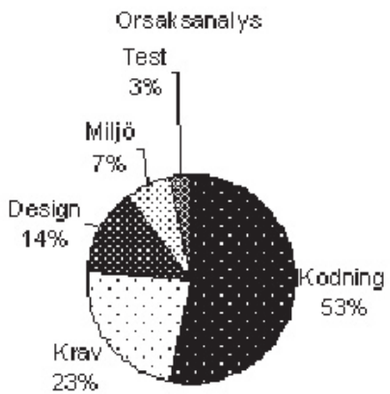
Det är mycket viktigt att alla diagram du presenterar sätts i sitt sammanhang.

23.5 Analys i efterhand

Det är intressant att efter avslutat projekt analysera testarbetet med mål att lära oss vad vi kan göra bättre nästa gång. Några av de viktigaste frågorna vi kan ställa oss är:

- Hur fungerade vårt upplägg av tester rent praktiskt?
- Hur uppfylldes målet att hitta allvarliga fel i början och mindre allvarliga mot slutet?
- Hittade vi rätt sorts fel i rätt testnivå? Om inte, vad berodde det på?
- Vilka var orsakerna till de fel vi hittade, berodde det på krav, design, kodning eller faktorer hos test?
- På vilka sätt kan vi göra ett bättre arbete nästa gång?

Påfallande ofta finns det ett stort motstånd mot att analysera vad som gått fel. Orsaken till detta kan vara en ovilja att förändra eller en misstro mot att enskilda personer ska pekas ut. Detta är synd då vi missar chansen att hitta våra svaga punkter och åtgärda dem inför nästa projekt.



Figur 23.4: Analys av orsak till avvikelser. Det är intressant att efter avslutat projekt analysera testarbetet med mål att lära oss vad vi kan göra bättre nästa gång.

	Klass 1 - Stoppande	Klass 2 - Allvarligt	Klass 3 - Mindre allvarligt	Klass 4 - Skönhetsfel	Klass 5 - Ogiltigt	Summa för nivå	Totalt antal fel kvar	Felupptäckning
Programintegrationstest	2	6	0	0	2	8	111	7%
Systemtest	6	17	13	14	0	50	103	49%
Systemintegrationstest	3	12	1	2	0	18	53	34%
Acceptanstest	0	7	12	6	0	25	35	71%
Produktion	0	5	1	4	0	10		
Totalt antal hittade fel						111		

Figur 23.5: Tabell över felupptäckning för olika testnivåer. Det är intressant att se hur många fel som slinker igenom varje testnivå, speciellt då det i medeltal kostar mer ju senare felen upptäcks. Vi kan aldrig nå 100 procents felupptäckning i varje nivå men vi kan säkert bli bättre och därmed tjäna mer pengar.

Analys av hur många fel vi hittat per testnivå ger en siffra i procent för felupptäckning. Ordet felupptäckning kommer från engelskans *defect detection percentage* vilket egentligen ska översättas med det längre ordet felupptäckningsprocent. Beteckningen står för hur stor del av de fel som finns kvar som du upptäcker i en viss testnivå eller fas. Eftersom det är dyrare att hitta och korrigera fel ju senare de upptäcks vill vi ha ett högt procenttal för varje nivå. Speciellt intressant är att räkna ut felupptäckning för projektet som helhet. Det kan du göra genom att efter en viss tidsperiod av produktion, antingen i antal månader eller innan nästa release, summera hur många fel som inträffat. För vårt exempel är detta:

$$\frac{\text{totalt antal fel i test}}{\text{totalt antal fel i test plus sex månaders produktion}} = \frac{8+50+18+25}{8+50+18+25+10} = 91\%$$

Figur 23.6: Uträkning av felupptäckningsprocent för vårt arbete innan produktion. Då vi hittar ytterligare tio fel de första sex månaderna efter att systemet testats klart betyder det att vi upptäckt 101 av 111 fel vilket är 91 procent. En mer rättvisande beräkning tar även med klassning av allvarlighetsgrad som en parameter.

Ett mätbart förbättringsmål kan vara att minska antalet fel som slinker igenom till produktion så att felupptäckningsprocenten går upp mot 95 procent. Även här kan det vara intressant att i beräkningarna ta med hur allvarliga felen är genom att bara räkna på allvarliga fel eller att vikta dem olika.

24 När är vi klara med testerna?

Det finns inget enkelt sätt att avgöra när ett system är färdigtestat. De flesta författare är överens om att det inte finns något enskilt kriterium som vi kan utgå från för att besluta att vi är klara. Här presenteras några varianter på synsätt.

24.1 Fem kriterier för att avsluta testerna

Enligt Lee Copeland finns det fem grundläggande kriterier som tillsammans brukar användas för att besluta när du ska sluta testa.[103] Dessa är:

1. Vi har uppnått de mål för täckning som vi definierat i strategin.
2. Antalet upptäckta avvikelser är färre än det gränsvärde vi definierat.
3. Kostnaden för att hitta fler fel är större än uppskattad förlust på grund av kvarvarande fel.
4. Projektteamet drar tillsammans slutsatsen att produkten är redo att släppas.
5. Den som bestämmer ger order om att produktionssätta.

Det finns en mängd svagheter med dessa kriterier tagna var för sig. Glenford Myers menar att målet med att nå täckning kan riskera att styra testarna till att skriva färre eller sämre testfall bara för att kunna hinna med att köra allt.[104] Att vi inte upptäcker fler avvikelser kan betyda att vi inte tes-

[103] Copeland, Lee [2003]: A Practitioner´s Guide to Software Test Design, s 236
[104] Myers, Glenford [1979]: The Art of Software Testing, s 122- 128

tar på rätt sätt eller att de bästa testarna är på semester, det betyder inte säkert att det inte finns fler felaktigheter kvar. Att kostnaden är större än nyttan med fler testfall är en högst subjektiv värdering som säkert inte testaren är utrustad att göra, det är mer en affärsfråga. Teamets konsensus kan vara ett relativt sett bättre mått då vi här diskuterar med utvecklare, testare och verksamhetskunniga tillsammans. Det sista kriteriet är en deadline som beslutas utanför testarnas domäner och har egentligen inget att göra med hur bra vi testat utan bygger på en ren affärsbedömning av att vi måste släppa produkten vid ett visst datum. Det är vanligt att flera av ovanstående kriterier används för att tillsammans utgöra stoppkriterium för test.

24.1 Tillräckligt bra kvalitet

James Bach beskriver ett synsätt han kallar *good enough quality* som han sammanfattar i följande fyra punkter som alla måste vara uppfyllda.[105] En sak har tillräckligt hög kvalitet om:

1. Den har tillräckligt med fördelar.
2. Den saknar kritiska problem.
3. Fördelarna överväger tillräckligt mycket mot nackdelarna.
4. Vidare tester och förbättringar skulle, i den rådande situationen och med alla delar i åtanke, göra mer skada än nytta.

För att komma fram till ovanstående kan vi förklara varje del lite mer i detalj. Ställ följande frågor:

1. Vilka specifika fördelar har vår produkt? Hur stor är sannolikheten att någon av våra intressenter kommer att använda en viss fördel? Hur viktig är varje fördel? Vilka delar är kritiska? Är alla fördelar sammantaget tillräckligt bra för våra intressenter?
2. Vilka potentiella problem har vår produkt? Hur stor är sannolikheten att någon av våra intressenter kommer att råka ut för ett visst problem? Hur skadligt kan varje problem vara? Vilka problem är helt oacceptabla? Är alla problem sammantaget för mycket för att våra intressenter ska vara nöjda?
3. Har produkten tillräckligt mycket fördelar så att de nackdelar som eventuellt finns är tillräckligt få? Hur bra måste produkten vara för att accepteras av kunderna?
4. På vilka sätt skulle vi kunna förbättra produkten? Vad skulle detta innebära för kostnader? Finns det möjlighet att leverera nu och leverera förbättringar senare? Exakt vilka fördelar skulle vi få av att förbättra? Vilka problem skulle vi kunna få om vi arbetade vidare exempelvis förstöra sådant som fungerar, ta resurser från andra projekt?

Slutsatsen av ovanstående förhållningssätt är att vi måste använda flera olika kriterier tillsammans för att få ett bra beslut. Dessutom är det oftast inte testarna som beslutar om något ska släppas, vi informerar om vad vi kommit fram till utifrån vårt arbete och lämnar till projektledare eller styrgrupp att ta beslut om release. Däremot kommer du som testare säkerligen att få frågan om vad du anser om kvaliteten. Då är det bäst att du har ett svar som är väl underbyggt.

[105] Bach, James [1997]: Good Enough Quality: Beyond the Buzzword, Computer, 8/97

25 Lösningsförslag till övningar

25.1 Ekvivalensgrupper

I detta fall har jag genom diskussion med kravställaren kommit fram till att utbetalningssätt väljs i en lista (drop down-meny), och därför finns det endast fyra värden att testa. Övriga fält är manuellt ifyllda och har en del grundläggande felkontroller av format.

Även om jag antar att programtest är kört på max/min och felvärden gör jag ändå en del stickprov i systemtest. Att testa samtliga felaktiga värden skulle ta mycket extra tid – detta ska redan vara utfört! Listan på nästa sida är vad jag bedömer som en rimlig nivå för systemtestfall.

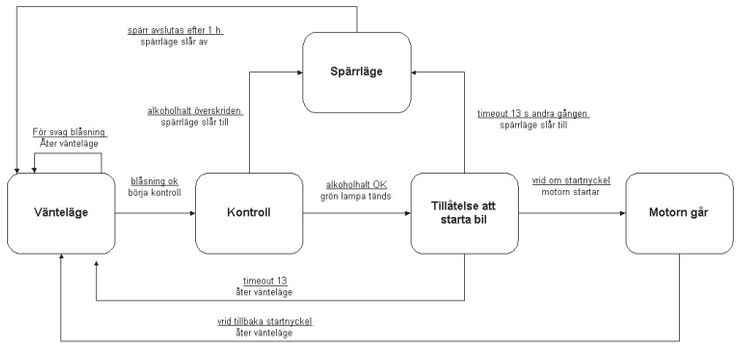
Beskrivning				1	2	3	4	5	6	7
			Prio	1	2	2	2	2	2	3
Parameter	Grupper och gränsvärde	Värden								
Efternamn	Giltiga	a-z, A-Z		X						X
		å,ä,ö,Å,Ä,Ö			X					X
		de, af, dubbelnamn med mellanslag				X				X
		bindestreck, accenter, utländska namn som O´Connor					X			
		1-20 tecken		X			X			X
		1 tecken A-Ö (efternamnet Å finns i telefonkatalogen!) (min)			X					
		20 tecken (max)				X				
	Ogiltiga	tomt fält						X		
		>20 tecken							X	
		21 tecken								
		ett tecken som är blanksteg								
		ej tillåtna tecken ex siffror, @, %								
Utbetal-nings-sätt	Giltiga	Personkonto		X			X	X	X	
		Bankkonto			X					X
		Utbetalningskort				X				
	Ogiltiga	Val saknas								
Belopp	Giltiga	1-999 999 999 (kan du ha mellanslag eller . som avdelare?		X			X	X	X	X
		1 (finns det annan högre gräns i regelverket ex 50 kr?) (min)			X					
		999 999 999 (max)				X				
	Ogiltiga	decimaltal								
		icke-numeriska tecken ex bokstäver, räkneoperatorer								
		<1								
		0								
Skatt	Giltiga	0-100		X			X	X	X	X
		0			X					
		100				X				
	Ogiltiga	>100 %								
		< 0 %								
		decimaltal								
		icke-numeriska tecken								
		tomt fält								
		blanksteg								

Figur 25.1: Lösningsförslag för övning att ta fram ekvivalensgrupper. Nedanstående lista är vad jag bedömer som en rimlig nivå för systemtestfall. Det slutliga utseendet bestäms av din kunskap om systemets logik och hur detaljerat du vill testa.

25.2 Tillståndsgraf

Så här kan lösningen för alkolåset se ut. Observera att endast de giltiga vägarna ritats ut. En komplett graf innehåller även alla *otillåtna* händelser. Så

från vänteläget ska det finnas ytterligare en pil som går tillbaka in i vänteläge igen där det står – Vrid om nyckel/Inget händer. Om vi ska vara riktigt detaljerade så kan vi undra om det inte ska finnas ett extra läge där låset är på och nyckeln omvriden – innan nyckeln vrids tillbaka går det inte att blåsa. På samma sätt har de övriga tillstånden som exempelvis *kontroll* och *spärrläge* pilar som går tillbaka in i sig själva för händelserna vrid om nyckel eller blås.



Figur 25.2: Lösningsförslag för övning att ta fram ekvivalensgrupper. Nedanstående lista är vad jag bedömer som en rimlig nivå för systemtestfall. Det slutliga utseendet bestäms av din kunskap om systemets logik och hur detaljerat du vill testa.

Nr	Start	Händelse	Svar	Slut
1	Vänteläge	Blåsning ok	Gå vidare till kontroll av utandningsluft	Kontroll
2	Vänteläge	Blåsning för svag	Åter till vänteläge	Vänteläge
3	Kontroll	Alkotest visar på för hög alkoholhalt	Röd lampa tänds. Spärrläge timer sätts till 1 H	Spärrläge
4	Kontroll	Alkotest godkänd	Grön lampa tänds. Startspärren öppen	Tillåtelse att starta bil
5	Tillåtelse att starta bil	Timeout 13 sek	Åter till vänteläge. Registrera misslyckat startförsök	Vänteläge
6	Tillåtelse att starta bil	Vrid om startnyckeln	Motorn startar	Motorn går
7	Tillåtelse att starta bil	Timeout 13 sek för andra gången i rad	Röd lampa tänds. Spärrläge timer sätts till 1 H	Spärrläge
8	Motorn går	Vrid tillbaka startnyckel	Motorn stängs av	Vänteläge
9	Spärrläge	Timer för tidsspärr slår ifrån efter 1h	Åter vänteläge	Vänteläge

Figur 25.3: Lösningsförslag tabellöversikt övergångar för alkolås. Vi noterar detaljerat vad som leder till varje övergång och hur systemet svarar. Detta är viktig information för våra tester och är samtidigt en kvalitetssäkring av kraven.

I det här fallet är modellen ganska liten. Därför är det lämpligt att även rita en tabell för övergångspar och se till att täcka in alla par med testfallen.

Nästa sida visar en tabell över övergångspar och följande tre testfall täcker de flesta rader i denna. Ytterligare ett testfall behövs för att täcka in de sista raderna 7,11,13.

Tänk på att varje grundtestfall måste köras flera gånger för att testa både normala värden och gränsvärden!

Nr	Start	Steg 1	Steg 2
1	Vänteläge	Vänteläge	Vänteläge
2	Vänteläge	Vänteläge	Kontroll
3	Vänteläge	Kontroll	Tillåtelse att starta bil
4	Vänteläge	Kontroll	Spärr
5	Kontroll	Tillåtelse att starta bil	Motorn går
6	Kontroll	Tillåtelse att starta bil	Vänteläge
7	Kontroll	Tillåtelse att starta bil	Spärr
8	Tillåtelse att starta bil	Motorn går	Vänteläge
9	Tillåtelse att starta bil	Vänteläge	Vänteläge
10	Tillåtelse att starta bil	Vänteläge	Kontroll
11	Tillåtelse att starta bil	Spärr	Vänteläge
12	Kontroll	Spärr	Vänteläge
13	Spärr	Vänteläge	Vänteläge
14	Spärr	Vänteläge	Kontroll
15	Motorn går	Vänteläge	Vänteläge
16	Motorn går	Vänteläge	Kontroll

Figur 25.4: Lösningsförslag tabellöversikt övergångspar för alkolås. Att testa alla övergångspar ger en djupare test av systemet och hjälper oss att hitta fel som beror på kombinationer av övergångar.

Grundtestfall 1 – Happy day

Förutsättning: Du befinner dig i vänteläge, du är nykter.

Utförande:

1. Försök starta bilen utan att blåsa.
2. Försöket misslyckas.
3. Blås tillräckligt starkt.
4. Kontroll av luften sker.
5. Du får grönt ljus och kan vrida om nyckeln.
6. Starta bilen genom att vrida om nyckeln inom 15 sekunder.
7. Stäng av motorn genom att vrida tillbaka nyckeln.
8. Verifiera att det är så genom att försöka starta bilen utan att blåsa.

Resultat/sluttillstånd:

Du har startat och kört bilen med lyckat resultat. Den är nu åter i vänteläge.

Testtäckning: Detta testfall täcker raderna 3, 5, 8

Grundtestfall 2 – Happy night?

Förutsättning: Du befinner dig i vänteläge och är kraftigt berusad.

Utförande:

1. Blås tillräckligt starkt.
2. Kontroll av luften sker.
3. Du får rött ljus och får en spärr på en timme.
4. Efter en timme slår spärren ifrån.
5. Blås igen - fortfarande berusad.
6. Kontroll av luften sker.
7. Du får rött ljus och får en spärr på ytterligare en timme.
8. Efter en timme slår spärren ifrån och vänteläge inträder.
9. Nu blåser din kompis som är nykter.
10. Kontroll av luften sker.
11. Du får grönt ljus och kan vrida om nyckeln.
12. Men din kompis hittar inte rätt och timeouten är ett faktum. Ska ni behöva stanna ute hela natten?
13. Åter vänteläge.
14. Nu blåser din kompis som är nykter en gång till.
15. Kontroll av luften sker.
16. Ni får grönt ljus och kan vrida om nyckeln.
17. Denna gång vrider din kompis om nyckeln inom 15 sekunder.

18. Motorn startar.
19. Efter att ha parkerat säkert hemma vrids nyckeln tillbaka.
20. Motorn stannar.
21. Din fru/man undrar vad som står på och du vill gärna visa. Ta en klunk whisky och blås igen.
22. Luften kontrolleras och du får rött sken.
23. Startspärren på en timme slår till igen och det går inte att starta bilen hur mycket du än försöker.

Testtäckning: Detta testfall täcker förutom tidigare redan täckta rader även: 4, 6, 10, 12, 14

Grundtestfall 3: Blåsigt värre

Förutsättning: Systemet är i vänteläge och du är nykter men har dåliga lungor.

Utförande:

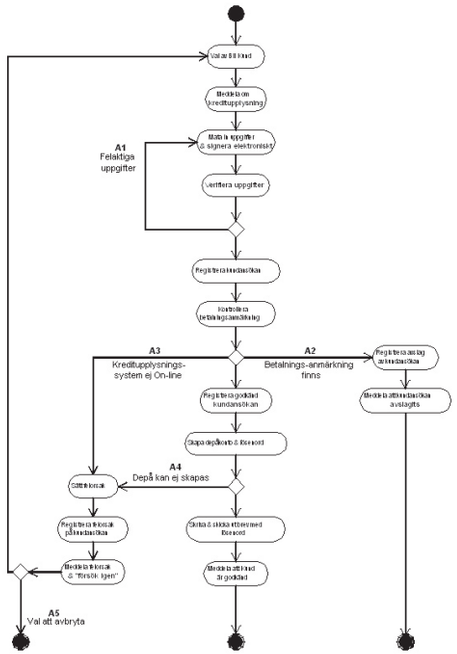
1. Blås för svagt.
2. Du hamnar åter i vänteläge.
3. Blås för svagt igen.
4. Du är fortfarande i vänteläge.
5. Blå tillräckligt starkt.
6. Luften kontrolleras.
7. Grön lampa.
8. Timeout då du inte startar i tid.
9. Blås igen men för svagt.
10. Blås igen och tillräckligt starkt denna gång.
11. Luften kontrolleras.
12. Du får grön lampa.
13. Starta bilen inom 15 sekunder.
14. Motorn går.
15. Stäng av motorn.

16. Åter vänteläge.
17. Blås igen men för svagt.
18. Åter vänteläge.

Testtäckning: Detta testfall täcker förutom tidigare redan täckta rader även: 1, 2, 9, 15

25.3 Användningsfall

Vi börjar med att ta fram aktivitetsdiagrammet.



Figur 25.5: Lösningsförslag till aktivitetsdiagram för kund aktiehandel. Vi har tre olika slutpunkter som alla har olika resultat. Endast i ett fall har en depå skapats, i de två andra har kunden antingen valt att avbryta eller har nekats att öppna en depå. För test är det extra viktigt att tydligt markera olika resultat för alternativflöden.

De styrande variablerna noteras i en tabell.

	Styrande variabler									Förväntat resultat
Scenario /TF-ID	Prio	Förnamn	Efter-namn	nr	Bank-kontonr	Elektr. sign.	Kund-system	Kredit-upplysning		
Ha	1	Eva	Marklund	651212-nnnn	nnnnnn	OK	Kund finns,	Ej anmärkn,		Depåkonto, lösenord och användarprofil => OK
/H-001					Löne-konto		Online	Online		
Hb	1	Johan	Turesson	780506-nnn	nnnnnn	OK	Kund finns,	Ej anmärkn,		Depåkonto, lösenord och användarprofil => OK
/H-002					Spar-konto		Online	Online		
A1a		Efva	Marklund	651212-nnnn	nnnnnn	OK	Kund finns,	Ej anmärkn,		Meddelande om att personnummer ej stämmer överens.
/A1-001					Löne-konto		Online	Online		Ingen användarprofil, depåkonto eller lösen skapas.
A1b		Johan	Turesson	780506-nnn	annan persons kontonr	OK	Kund finns,	Ej anmärkn,		Meddelande om att angivet kontonummer ej tillhör Potentiella Kunden.
/A1-002							Online	Online		Ingen användarprofil, depåkonto eller lösen skapas.
A1c		Johan	Turesson	780506-nnn	felaktigt kontonr	OK	Kund finns,	Ej anmärkn,		Meddelande om att kontonummer ej stämmer finns.
/A1-003							Online	Online		Ingen användarprofil, depåkonto eller lösen skapas.
A1-d		Johan	Turesson	780506-nnn	nnnnnn	Fel	Kund finns,	Ej anmärkn,		Meddelande om felaktig elektronisk signatur.
/A1-004					Spar-konto		Online	Online		Ingen användarprofil, depåkonto eller lösen skapas.
A2		Christin	Alvunge	740303-nnn	nnnnnn	OK	Kund finns,	Anmärkn finns,		Meddelande om att ansökan avslås maa anmärkningen.
/A2-005					Löne-konto		Online	Online		Ingen användarprofil, depåkonto eller lösen skapas.
A3		Christin	Alvunge	740303-nnn	nnnnnn	OK	Kund finns,	EJ		Meddelande om att ansökan ej kan behandlas för tillfället.
/A3-006					Löne-konto		Online	Online		Ingen användarprofil, depåkonto eller lösen skapas.

Figur 25.6: Lösningsförslag styrande variabler "bli kund aktiehandel". Vi har valt att ta med samtliga variabler som finns uppräknade i användningsfallet även om man kan diskutera om de alla är "styrande".

När variablerna som påverkar flödets förväntade resultat kommer upp i tabellen och scenarios börjar ta form uppstår ofta ett antal obesvarade frågor och funderingar på om det som står beskrivet är korrekt tänkt. Här är några exempel på frågor som framkom då vi tog fram aktivitetsdiagram och styrande variabler:

- Varför är namn, personnummer etc styrande variabler? Borde inte systemet redan ha dessa då det krävs att kunden är inloggad på sin internetbank för att kunna lämna ansökan? Osäkra förutsättningar.
- Kan Potentiell Kund endast avbryta vid steg A3-4 och A4-4?
- Om depåkonto inte kan skapas, hur gör Aktiehandelssystemet med den registrerade och godkända kundansökan? För kundens skull borde den väl sparas så vi kan meddela resultat senare?
- Om depåkonto redan finns, hur ska Aktiehandelssystemet agera?
- Om Kundsystemet ej är online, hur kan den Potentiella Kunden verifieras?
- Hur långt tillbaka i tiden ska Aktiehandelssystemet ta hänsyn till eventuell betalningsanmärkning?
- Om vi gör kreditupplysning varje gång kunden ansöker medför uppmaningen om att kunden ska ansöka igen en ökad kostnad för oss själva. (När det handlar om pengar brukar folk reagera).

Frågorna tas upp tillsammans med kravställaren och därefter revideras kraven med förtydliganden och ändringar.

Vi har plockat fram följande förslag till ett av grundtestfallen.

Id	Grundtestfall H Bli Kund - Skapa användarprofil, depåkonto och lösenord				
Syfte	Skapa en användarprofil med depåkonto och lösenord för en användare som är kund i banken sedan tidigare				
System	AKTIE	Status	Utkast	Aktör	Potentiell Kund
Författare	ANB026	Prioritet	Hög		
Testnivå	ST/AT	Datum	6/23/05		
Förutsättningar	- Den Potentiella Kunden är redan Kund i banken med aktuella konton				
	- Den Potentiella Kunden är inloggad på internetbanken				
	- Aktiehandelssystemet är uppe				
	- Kundsystemet är uppe				
	- Kredituppluysningens system är uppe				
	- Inga betalningsanmärkningar finns				
Testdata	Se tabellen styrande variabler för AF Bli Kund.				
	Aktuella Scenario/Testfall: Ha/H001 & Hb/H002				
Steg	Beskrivning	Förväntat resultat			
1.	Välj att "Bli Kund"	Fönster visas för att registrera personlig information			
2.	Registrera förnamn, efternamn, personnummer och bankkonto. Välj att skicka uppgifterna.	Dialog visas för att elektroniskt signera kundansökan			
3.	Registrera aktuell elektronisk signatur. Välj att gå vidare	Dialogen stängs och fönster visas med information om att ansökan behandlas			
4.	-	När ansökan har blivit godkänd av Aktiehandelssystemet visas nytt fönster med information om att ansökan är godkänd, depåkonto är upplagt och lösenord skickas per brev			
5.	Välj "Startsida"	Fönster med internetbankens första sida visas			
Kontroll av resultat efter att alla steg är körda					
1.	Kontrollera att depåkonto är skapat för Kunden				
2.	Kontrollera att rätt bankkonto är kopplat till depåkontot				
3.	Säkerställ att en utdatafil har genererats med Kundens lösenord	Utdataformatet verifieras i annat testfall			
4.	Kontrollera att det går att logga in på Aktiehandelssystemet med genererat lösenord	Lösenord hämtas via teststödapplikationen "Teststödet"			

Figur 25.7: Ett grundtestfall för "bli kund aktiehandel". Vi har valt huvudflödet och beskrivit testfallet detaljerat så att en verksamhetsperson kan delta i utförandet med lite stöd.

25.4 Beslutstabell

Jag hittar sex regler som finns insmugna i fältbeskrivningen, dessa finns nedskrivna i tabellen.

Jag väljer att testa varje regel genom att lämna ett obligatoriskt fält tomt i taget. Det går relativt snabbt att köra eftersom det är en dialog som ger mig direkt återkoppling. För regel 3 t ex skulle jag fylla i Eurobond som instrumentklass och sen lämna Fi-grupp tomt, men clearingmode och konto ifyllt. Nästa steg är att fylla i Fi-grupp men ta bort Clearingmode o s v. På så sätt kontrollerar jag att fältkontrollen fungerar för ett fält i taget.

Fältnamn	Värdemängd	Regel0	Regel1	Regel2	Regel3	Regel4	Regel5
Kortnamn	Ej editerbart						
Kundnummer	Ej editerbart						
Fi-grupp	01,02,03	Obligatoriskt	01	Obligatoriskt	Obligatoriskt	Obligatoriskt	Obligatoriskt
Clearingmode	KASS, CEDE, EUCL		Obligatoriskt	Obligatoriskt	Obligatoriskt	Obligatoriskt	Obligatoriskt
Instrumentklass	Eurobond, blankt			Eurobond			
Affärstyp	AVM, BSP, DEP, EMU, DFO				Affärstyp ifylld	Obligatoriskt	Obligatoriskt
Land	Landkod std					Landkod ifylld	Obligatoriskt
Valuta	Valutakod std						Valutakod ifylld
Konto	Euroclear, Kassevere, Cedel	Obligatoriskt	Obligatoriskt	Obligatoriskt	Obligatoriskt	Obligatoriskt	Obligatoriskt

Figur 25.8: Lösningsförslag beslutstabell för fältkontroll. Jag hittar sex regler som finns insmugna i fältbeskrivningen, dessa finns nedskrivna i tabellen. Jag väljer att testa varje regel genom att lämna ett obligatoriskt fält tomt i taget.

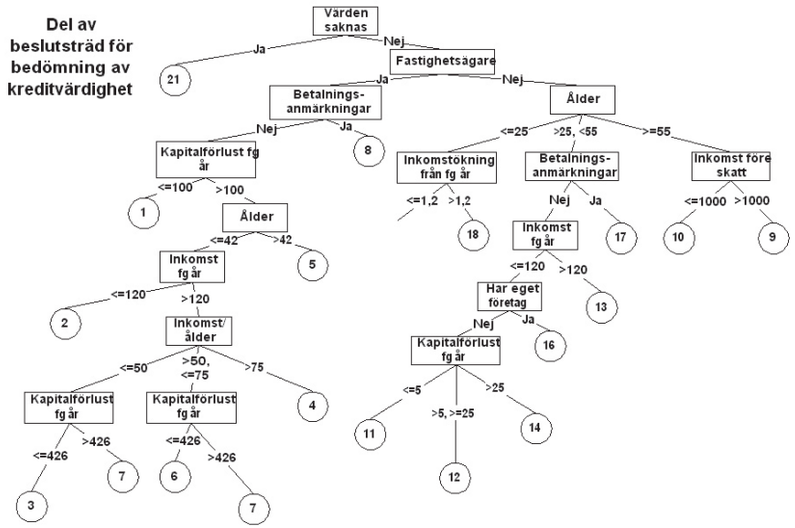
25.5 Beslutsträd

Tips: Skriv in regelverket i Excel och dölj rader och kolumner allt eftersom de ritas in i trädet.

1. Trädets första villkor är felhanteringen som finns i regel 21.
2. Nästa uppdelning gjordes på om personen var fastighetsägare eller ej

– denna parameter fanns med i alla de regler som fanns kvar då regel 21 försvunnit.

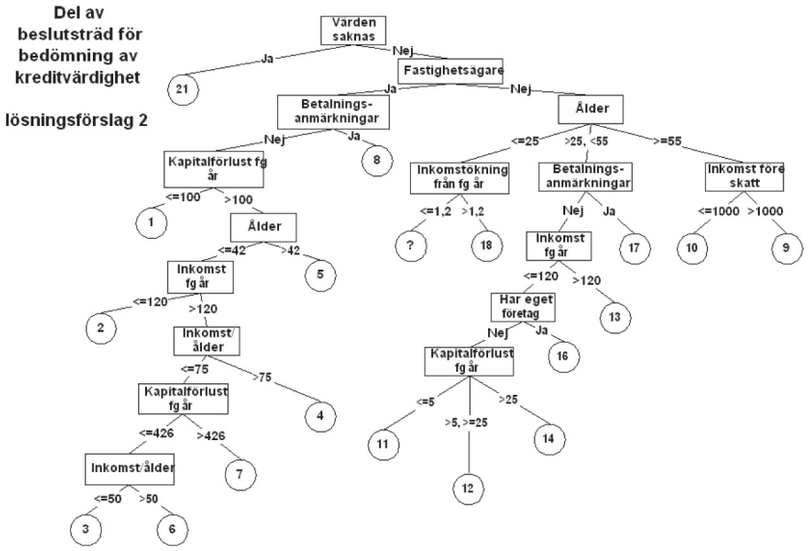
3. För de som var fastighetsägare var nästa parameter om det fanns betalningsanmärkningar eller ej.
4. Nästa parameter var kapitalförlust o s v.
5. Då regelverket är komplext får vi antingen dela upp vissa regler till flera olika löv eller ta upp parametern kapitalförlust på flera ställen i trädet. Detta leder till att reglerna 7 och 13 finns med i flera löv i lösning 1. En regel saknas vid villkoret *Inkomstökning fg år <=1.2*



Figur 25.9: Lösningsförslag 1 beslutsträd kreditregelverk. Då regelverket är komplext får vi antingen dela upp vissa regler till flera olika löv eller ta upp parametern kapitalförlust på flera ställen i trädet. Detta leder till att reglerna 7 och 13 finns med i flera löv i lösning 1.

Lösningsförslag 2

Här har vi gjort trädet djupare genom att istället ha med samma parameter flera gånger. Vi får då färre löv. Det medför att vi får färre testfall. Här har vi lagt till den regel som saknas med ett frågetecken istället för regelnummer.



Figur 25.10: Lösningsförslag 2 beslutsträd kreditregelverk. Här har vi gjort trädet djupare genom att istället ha med samma parameter flera gånger. Vi får då färre löv. Det medför att vi får färre testfall. Här har vi lagt till den regel som saknas med ett frågetecken istället för regelnummer.

Testfallen byggs sedan utifrån trädet.

För att testa varje löv sätter vi ihop minst två testfall, ett normalvärde och ett gränsvärde. Hur många som krävs beror på hur koden ser ut. I det riktiga projektet användes två till fyra testfall per regel för att få en bra täckning.

Testfall där något värde saknas testas i regel 21. Du kan läsa mer om hur många testfall du bör ha i nästa teknik grundläggande jämförelser. Den säger att varje ingående variabel ska få bestämma någon gång. I vårt fall betyder det att varje värde som ingår i felhanteringen ska saknas ett i taget medan de andra finns.

Testdata skapas sen med en rad i datafilen för varje testfall. Facit är att testfallet ska få träff på rätt regel och att rätt ratingvärde 1-5 tilldelas.

25.6 Grundläggande jämförelser

25.6.1 Lösningsförslag

	Beskrivning	Test-fall 1	Test-fall 2	Test-fall 3	Test-fall 4	Test-fall 5
R1.1 (0)	Ålder 18	X		X	X	X
R1.2 (1)	Ålder < 18		X			
R2.1 (0 1 1)	Ålder >40 är falsk så regeln blir falsk		X			
R2.2 (1 0 1)	Lön < 20k är falsk så regeln blir falsk	X				
R2.3 (1 1 0)	Saknar bet anmärkning är falsk så regeln blir falsk			X		
R2.4 (1 1 1)	Alla villkor sanna så regeln blir sann				X	X
R3.1 (1 0 0 0)	Guldmedlem gör att extra avgift på 500 läggs på	X				
R3.2 (0 1 0 0)	Elitspelare gör att extra avgift på 500 läggs på		X			
R3.3 (0 0 1 0)	Bollprenumeration gör att extra avgift på 500 läggs på			X		
R3.4 (0 0 0 1)	Racketservice gör att extra avgift på 500 läggs på				X	
R3.5 (0 0 0 0)	Inga specialvärden ger 0 i ändring					X
	Avgift	2500	0	2500	2000	1500

Figur 25.11: Lösningsförslag grundläggande jämförelser. Fyra variabler ger 4+1 testfall om vi först testar en variabel i taget som sann och till sist alla variablerna falska samtidigt. Vi kan kombinera testvillkoren med redan existerande testfall då reglerna är oberoende av varandra.

25.6.2 Diskussion om uteblivet testfall

I detta fall hoppar vi över villkoret där en medlem har flera av villkoren elitspelare, guldmedlem, bollprenumeration eller racketservice då dessa var för sig redan har bestämt att 500 kronor läggs på avgiften. Men vad händer om utvecklaren tolkar kravet som att elitspelare och guldmedlem medför 500 kronor i tillägg var och en? Detta fel missar vi om vi hoppar över det testfallet. Eller kanske ska det vara så och kraven är fel?

Tänk på att tekniken bara är vägledande och att den bara hittar vissa typer av fel. Tekniker är bra att ha men det är din kunskap som är avgörande för hur bra testerna blir.

26 Ordlista

Attackmönster. James Whittakers bidrag till hur vi tar fram bra testfall. Tekniskt fokus.

Avvikelse. Ett testresultat som skiljer sig från det förväntade.

Black box test. Vi betraktar systemet som en svart låda och bryr oss enbart om indata och utdata och inte exakt vad som händer inuti systemet.

Boolsk algebra. Notationssätt som togs fram av matematikern Boole som hjälp att lösa logiska problem. Behandlar dataset och operationer på dessa samt logiska operationer. Specifikt de logiska operatorerna AND, OR, NOT som är grunden i all databehandling.

Context-Driven School. Sammanslutning som arbetar utifrån förutsättningen att alla testinsatser beror på sammanhanget, det finns ingen universallösning.

Defekt. Se avvikelse.

Dialog. Grafiskt användargränssnitt där användaren direkt kan interagera med systemet.

Domäntest. Ekvivalensgruppering där flera variabler samverkar och därför måste testas tillsammans.

Ekvivalensgrupp. Gruppering av data eller testfall som antas testa samma saker. Principen är att endast ett värde i samma grupp behöver testas.

Exploratory testing. Se utforskande testning.

Felupptäckning. Hur stor del av de fel som finns upptäcker du i en viss testnivå eller fas. Kallas även DDP - *defect detection percentage*.

Genomförandeplan. Ett schema som beskriver hur testerna ska utföras.

Grundtestfall. Testfall där vi ännu inte har brytt oss om att beakta testdata i detalj.

Gränsvärden. Vi utgår från ekvivalensgrupperna och testar de värden som ligger i utkanten på varje grupp, exemplevis max och min.

Happy day. Beteckning på enkla inledande tester. Kallas även *smoke test* och *health check*. En av de få termer vi inte översatt till svenska ännu.

Heuristik. Kortfattade riktlinjer som ska hjälpa oss att angripa ett problem.

Icke funktionell testning. Samlingsbegrepp på tester som omfattar delar som inte räknas som direkta funktioner, exempelvis användbarhet, prestanda. Kallas även kvalitetsegenskaper.

ISTQB. International Software Testing Qualifications Board, ansvarar för internationell terminologi och certifiering.

Komponent. En samling kod som realiserar en funktion.

Kravanalytiker. Roll som finns i vissa utvecklingsprocesser. Tar emot och analyserar de krav som tagits fram.

Modell. En förenkling av en verklig situation med syfte att hjälpa oss att lösa problem.

Orakel. Den källa som ger oss förväntat resultat av en test.

Persona. En precis beskrivning av användaren, vad han vill åstadkomma och varför. Metoden vänder sig främst till designers och arkitekter men är även användbar för testdesign.

Popper, Karl. Av mig utsedd till testarnas egen filosof. Författare till bland annat den vetenskapsfilosofiska boken *Conjectures and Refutations*. Popper beskriver hur vår kunskap om vetenskapen växer fram genom att vi utsätter våra hypoteser för kritiska tester. På samma sätt utsätter vi testare programvaran för våra egna kritiska tester. Vi kan aldrig veta säkert om något är korrekt men ju fler avancerade tester vi lyckas genomföra desto säkrare blir vår uppfattning om att det fungerar.

Programvara. Används istället för ordet mjukvara som är en översättning av engelskans software.

Scenario. Väg genom ett användningsfall.

Skriptad testning. Tester som utförs med hjälp av tidigare nedskrivna instruktioner. Motsatsen till utforskande testning.

Statisk test. Utförs utan att koden exekveras, t ex granskning eller analys av dokument eller kod.

Strukturell test. Fokus är på vad som sker i koden, rätt anrop till rätt tabell, viktigt men inte tillräckligt att täcka all kod med testerna.

Taxonomi. Ett sätt att klassificera saker genom att gruppera dem efter liknande egenskaper.

Tekniska tester. Jag använder beteckningen tekniska för att markera att testerna ligger närmare koden än den vanliga användarens gränssnitt.

Testfall. En samling instruktioner som beskriver hur en test ska utföras.

Testkedja. En följd av sammanhängande testfall som körs i en viss ordning.

Testpaket. Ett sätt att praktiskt samla testfall som t ex ska köras samtidigt eller av samma person.

Unified process. Förkortat UP, skapades av Ivar Jacobson. En kommersiell variant kallas RUP där R står för företaget Rational.

Utforskande testning. Utforskande testning är samtidigt lärande, testdesign och testutförande i motsats till skriptad testning.

White box test. Se strukturell test.

27 Lästips

27.1 Test

Black, Rex : Managing Software Testing och Critical Testing Processes
Praktiska tips för dig som vill veta mer om hur testledning går till.

Copeland, Lee : A Practitioners Guide to Test Design
En alldeles utmärkt bok som i detalj går igenom många av de viktigaste grundläggande teknikerna för testdesign. Lättläst och med många bra exempel. Den bästa boken på engelska om testdesign.

Gilb, Tom, Dorothy Graham: Software Inspection
En grundlig genomgång av granskningstekniker för programvaruutveckling.

Kaner, Cem: Testing Computer Software
Denna bok marknadsförs med devisen ”Mest sålda testbok genom tiderna”. Det är uppenbart att författarna har en gedigen erfarenhet av test, de har fyllt boken med en massa smått och gott. På dryga 500 sidor beskrivs ett brett spektrum av test från detaljer i testdesign till felhantering och testledning. Köp den och ha som uppslagsbok, läs ett nytt avsnitt för att få idéer.

Kaner, Cem, James Bach, Brett Petticord: Lessons Learned in Software Testing

293 tips på 263 sidor – en blandning av kortare och lite längre tips. Boken bygger helt på författarnas egna upplevelser. Att läsa den är som att ha ett samtal med några av världens bästa testare och det är en inspirationskälla för mig i mitt arbete med test. Se inte varje mening som en absolut sanning utan som en inspiration till nytt tänkande. Det är ett bra komplement till din övriga referenslitteratur då denna bok ifrågasätter många gamla sanningar. Boken kommer från the Context-Driven School som arbetar efter det faktum att vad som är bra är situationsberoende.

Kit, Edward: Software Testing in the Real World

En liten nätt bok på 240 sidor som handlar om förbättring av testprocessen. Den är lättläst och översiktlig och utgår från en variant av V-modellen som kallas "The Dotted-U Model". Modellen går ut på att visa hur varje del i kravställning och utveckling har en motsvarande aktivitet i test. Boken passar bra som introduktion för dig som vill lära dig grunderna inom test men den är även ett bra uppslagsverk för den mer erfarne testaren. Rekommenderas till ditt personliga referensbibliotek.

Myers, Glenford: The Art of Software Testing

Redan 1979 skrevs denna då banbrytande bok om test. Det har blivit något av en kultbok som ofta refereras till i modernare litteratur – det är faktiskt svårt att hitta en nyare bok om test som INTE har en referens till denna! Mycket av det som skrevs då är fortfarande lika aktuellt även om presentationen av det är lite tunt jämfört med den flora av litteratur som finns på området idag. Kul att ha hemma men behöver kompletteras med nyare litteratur.

Pol, Martin et al.: Software Testing – A Guide to the TMAP Approach

Denna bok ligger till grund för den teststandard som används i bland annat Holland, Belgien och Tyskland. Det är ett gediget uppslagsverk med instruktioner för allt från hur du skapar din teststrategi till hur du designar bra testfall. Mycket fokus på administration och dokumentaton. Har du certifierat dig enligt ISEB-standarden kommer du att känna igen dig i det mesta. Fördelen med TMAP är att du även kan använda boken som uppslagsverk då du vill gräva dig ner i varje enskild del. Innehåller ett kapitel om testdesigntekniker.

TPI - Test Process Improvement

Ett måste i varje seriös testspecialists bokhylla. En praktisk beskrivning av hur du stegvis förbättrar testprocessen. Själv har jag den som inspirationskälla till nästa aktivitet på företaget och samtidigt min egen utbildningsplan. Boken innehåller en beskrivning av olika testområden och vad som måste uppfyllas inom varje område för att nå en viss nivå. Däremot så är det ingen lärobok inom test.

Whittaker, James: How to Break Software

Praktiska tips för hur du systematiskt hittar de vanligaste felen i programvara. Fokus är på robusthet och tillvägagångssättet bygger på ingående kunskap om utveckling och programspråk, och författaren går rakt på teknikerna. Föredömligt få sidor. Exempel som är lätta att följa.

Whittaker, James: How to Break Software Security

Som hans bok *How to Break Software* men med fokus på hur du hittar säkerhetsproblem i programvaran.

27.2 Övrig litteratur

27.2.1 Användbarhet

Cooper, Alan: The Inmates Are Running the Asylum

Varför blir de system vi bygger så svåra att använda. En bok som alla inom programvaruutveckling och beställare av IT-system borde läsa. Både rolig och viktig och dessutom lättläst.

Cooper, Alan: About Face 2.0

Grunderna i interaction design. Hur vi får användbara system. En bok med möjlighet att revolutionera vårt sätt att designa tekniska produkter.

Ottersten, Ingrid och Johan Berndtsson: Användbarhet i praktiken

Om användbarhet på svenska.

27.2.2 Kravhantering och process

Fowler, Martin, Kendall Scott: UML Distilled.

Introduktion till notationsstandarden UML.

Kruchten, Philippe: Rational Unified Process

Introduction till RUP, nu även på svenska.

Leffingwell, Dean, Don Widrig: Managing Software Requirements: A Use Case Driven Approach

Kravhantering med användningsfall, en av de bättre böckerna på marknaden.

Robertson, Suzanne, James Robertson: Mastering the Requirements Process

Kravhantering med användningsfall, en av de bättre böckerna på marknaden. Tar upp hela processen.

Schneider, Geri, Jason Winters: Applying Use Cases

Introduktion till användningsfall.

Weinberg, Gerald: Exploring Requirements: Quality Before Design

Hur vi tar fram krav med kvalitet.

27.2.3 Filosofi, psykologi och vetenskapsteori

de Bono, Edward: Lateral Thinking

Lättläst och kul om problemlösning. Bli bättre på att tänka utanför dina vanliga gränser och gör bättre avancerade testfall.

Levy, David A: Tools of Critical Thinking

Tänkvärt om fällor i vårt sätt att tänka. Egentligen en lärobok för psykologer men väl värd att läsa.

Popper, Karl: Conjectures and Refutations

Detta verk beskriver hur vetenskapen växer fram genom att vi först tar fram hypoteser och gissningar, sen utsätter vi dem för kritiska tester för att se om de håller. Samma tillvägagångssätt gäller för test av programvara där vi genom att ta fram riktigt bra testfall ser till att slutprodukten håller hög kvalitet. Ska du läsa någon filosof, läs Popper.

Weinberg, Gerald: An Introduction to General Systems Thinking

Nyttig bok som handlar om vetenskapsteori. Hur löser vi problem genom att bygga modeller och systematisera.

27.3 Internetresurser

www.howtobreaksoftware.com

-Här kan du ladda ner de senaste versionerna av *Holodeck* och *Canned Heat* som hjälper dig att simulera olika problemsituationer för test av robusthet.

www.model-based-testing.org

- Massor av länkar till intressanta artiklar om modeller som hjälper oss i test. Speciellt fokus på formella modeller och automatisering.

www.satisfice.com -

Har en stor samling läsvärda artiklar för läsning och nedladdning. Läs det senaste om *Exploratory Testing* och ladda ner mallar och gratisprogram som *allpairs.exe*.

www.context-driven-testing.com

- Information om detta angreppssätt.

www.sast.se

Sveriges största förening för test. Här hittar du information om konferenser, utbildning, litteraturrecensioner m m.

27.4 Standarder inom test

Nedan listas de mest spridda standarder som finns idag. Den mesta dokumentationen är på engelska.

- www.istqb.org

  - Här finns standarden för internationell certifiering med länkar till de olika ländernas nationella versioner.
- BS 7925-2:1998. Software Component Testing.
- DO-178B:1992. Software Considerations in Airborne Systems and Equipment Certification, Requirements and Technical Concepts for Aviation (RTCA SC167).
- IEEE Standard 610.12-1990
- IEEE 829:1998. Standard for Software Test Documentation.
- IEEE 1008:1993. Standard for Software Unit Testing.
- IEEE 1012:1986. Standard for Verification and Validation Plans
- IEEE 1028:1997. Standard for Software Reviews and Audits.
- IEEE 1044:1993. Standard Classification for Software Anomalies.
- IEEE 1219:1998. Software Maintenance.
- ISO 9000:2000. Quality Management Systems – Fundamentals and Vocabulary.
- ISO/IEC 9126-1:2001. Software Engineering – Software Product Quality – Part 1: Quality characteristics and sub-characteristics.
- ISO/IEC 12207:1995. Information Technology – Software Life Cycle Processes.
- ISO/IEC 14598-1:1996. Information Technology – Software Product Evaluation - Part 1: General Overview
- ISTQB kursplan [2005]: version 0.2 (finns en webadress istqb.org under standarder)

28 Referenser

Artiklar

Bach, James [1997]: Good Enough Quality: Beyond the Buzzword, Computer, 8/97

Bach, James [1999]: Heuristic Risk-Based Testing, Software Testing and Quality Engineering Magazine, 11/99

Bach, James [2005]: Rapid Testing, kursmaterial

Bach, James [2006]: Exploratory Testing Explained v 2.0

Boehm, B W [1976]: Software Engineering, IEEE Transactions on Computers, C-25

Chow, T.S. [1978]: Testing Software Design Modeled by Finite-State Machines, IEEE Transactions on Software Engineering 4

Fagan, Michael [1976]: Design and Code inspections to Reduce Errors in Program Development. IBS Systems Journal, 15(3)

Gardiner, Julie [2005]: EuroSTAR 2005

Gerrard, Paul [2002]: Risk-Based Test Reporting, StickyMinds.com

Gross, J. och J. Yellen [1998]: Graph Theory and its Applications

Grove Consultants [2002]: Test Practitioner, kursmaterial

Herzlich, Paul [1993]: W-model, www.evolutif.co.uk/default.asp?page=services/wmodel.html
www.testingeducation.org/BBST/BBST--IntroductiontoTestDesign.html
Kaner, Cem [2005]: groups.yahoo.com/group/software-testing/message/2880
Kwan, M-K. [1962]: Graphic Programming Using Odd and Even Points, Chinese Journal of Mathematics, Vol. 1
Mealy, GH [1955]: A Method for Synthesizing Sequential Circuits, Bell System Technical Journal 34 1045-1079
Moore, E.F. [1956] Gedanken Experiments on Sequential Machines In Automata Studies, Annals of Mathematical Studies nr 34 Princeton, NJ: Princeton University Press
Robinson, Harry [2005]: EuroSTAR Tutorial
Robinson, Harry [2005]: EuroSTAR Tutorial Model-Based Testing
Robinson, Harry, Steven Rosaria [1999]: Applying Models in your Testing Process, Intelligent Search Test Group, Microsoft Corporation
Royce, Dr Winston [1970]: Managing the Development of Large Software Systems
Windsor, Susan [2005]: EuroSTAR 2005 Keynote, www.context-driven-testing.com
Zimmerer, Peter [2005]: EuroSTAR 2005, presentation

Böcker och rapporter

Beck, Kent [2003]: Test-Driven Development
Beizer, Boris [1995]: Black Box Testing
Binder, Robert [2000]: Testing Object-Oriented Systems
Black, Rex [1999]: Managing the Testing Process
Cooper, Alan [2003]: About Face 2.0
Cooper, Alan [2004]: The Inmates are Running the Asylum
Copeland, Lee [2005]: A Practitioner's Guide to Software Test Design
De Voto, Bernard [1997]: The Journals of Lewis and Clark
Fowler, Martin, Kendall Scott [2003] :UML Distilled: A Brief Guide to the Standard Object Modeling Language (Object Technology S.)
Gilb, Tom, Dorothy Graham [1993]: Software Inspection
IEEE Standard 610.12-1990
ISTQB Kursplan version 0.2. www.sstb.se
Jacobson, Ivar, Grady Booch, James Rumbaugh [1999]: The Unified Software Development Process
Kaner, Cem, Jack Falk, Hung Nguyen [1999]: Testing Computer Software
Keay, John [1994]: The Permanent Book of Exploration
Kit, Ed [1995]: Software Testing in the Real World
Kruchten, Philippe [2002]: Rational Unified Process – En introduktion
Leffingwell, Dean, Don Widrig [2003]: Managing Software Requirements: A Use Case Approach
McConnell, Steve [2004]: Code Complete Second Edition
McGregor, John, David Sykes [2001]: A Practical Guide to Testing Object-Oriented Software
Myers, Glenford [1979]: The Art of Software Testing
Pol, Martin, Ruud Teunissen, Erik van Veenendaal [2002]: Software Testing – A Guide to the TMAP Approach
Robertson, Suzanne, James Robertson [1999]: Mastering the Requirements Process
Schneider, Geri, Jason Winters [2000]: Applying Use Cases
Schrage, M. [1996] Cultures of Prototyping i Winograd, T. (red) [1996] Bringing Design to Software, kap 10, New York: Addison-Wesley
Vijayaraghavan, Giri, Cem Kaner [2003]: "Bug taxonomies: Use them to generate better tests." www.testingeducation.org/a/tecrf.pdf
Weinberg, Gerald [1971]: The Psychology of Computer Programming

Weinberg, Gerald [2001]: An Introduction to General Systems Thinking, kap 1
Whittaker, James [2003]: How to Break Software
Whittaker, James [2003]: How to Break Software Security
Wiegers, Karl [2003]: Software Requirements, s 17

29 Index

A

Acceptanstest, 44-45, 49-51, 53-54, 217, 244, 247, 249, 266
agil, 49, 50, 58
agile, 49, 50, 58
allpairs, 202-205
användningsfall, 48,92,94, 97, 98, 101, 137,145-159,177-179, 232, 247, 250, 289
ASCII, 123
attackmönster, 93,97,102,220,287
avancerad testning, 94, 97, 109, 143, 154, 170, 199, 207, 218
avvikelse, 59,67,71,74-76,79,80-84, 86, 90, 91, 93, 115, 144, 145, 257-267

B

Bach, James, 27,35,38,56,61,99,112,202,221-222,270,292,299
Backus-Naur, 214
Beizer, Boris, 31,93,103,111,161
beslutstabell, 73, 93, 97, 98, 101, 126, 129
beslutsträd, 129,181-191, 283-286
Black Box-test, 92,94,98-101, 103, 111, 287
BNF, 214
Boehm, BW, 31-33
boktips, 291-295
boolsk algebra, 193-194, 287

C

Chow, TS, 165-299

Context-Driven School, 58, 65, 287, 292, 296, 300
Copeland, Allan, 96, 137, 293
CRUD, 101, 207-209, 238

D
datacykel, 207
dataflöde, 232, 236-238
defekt, 257, 287
domäntest, 97, 99, 101, 129-131, 190, 288
dual graf, 168-169
dynamisk teknik, 46-47, 89-90, 92-94

E
ekvivalensgrupp, 46, 97, 98, 101, 109, 115-135, 141, 235, 273, 288
enhetstest, 36, 39, 50, 51, 52, 193, 231-238
erfarenhetsbaserad, 220
exploratory testing, 21, 56, 61, 67, 87, 102, 288, 296

F
felgissning, 93, 97, 98, 102, 220
felrapport (se avvikelse)
frågemönster, 102, 226-229

G
genomförandeplan, 243, 252, 253-256, 288
glass box, 92, 94
granskning, 36, 42, 44, 45, 46, 89-91, 146, 226, 227, 290
Grey Box, 94, 95, 101
grundläggande jämförelser, 193-200, 235, 285, 286
grundtestfall, 109, 118, 140, 150, 152, 154, 164, 187, 188, 197, 208, 276
gränsvärde, 66, 70, 95, 98, 101, 109, 115-135, 141, 152, 154, 178, 188, 197, 212, 235, 269, 288

H
Happy Day, 150, 151, 254, 288
heuristik, 38, 66, 75, 83, 97, 102

I
icke-funktionell testning, 52, 95-96
inspektion, 30, 90
integrationstest, 45, 52, 95
ISTQB, 289, 296, 297

K
Kaner, Cem, 27, 31, 65, 80, 87, 98, 102, 257, 291, 296
kombinatorik, 93, 97, 101
kontrollflöde, 92, 95, 97, 99, 102, 224, 232-235, 238
kvalitetsegenskap, 89, 288

M
Mealy, GH, 161
modell, 108-109, 111-113
Moore, EF, 161, 168, 169
Myers, 103, 269, 292

O
orakel, 37, 38, 74, 75, 289

P
persona, 137, 289
Popper, Karl, 23, 29, 289, 295
prototyp, 143-144

Q
Q-pattern, 102, 226-229

R
risk, 46, 48, 56-57, 63, 78-79, 81-82, 86, 93, 97, 99, 102, 219-229
robusthetstest, 38, 96, 165, 170, 171, 225, 293, 295
RUP, 30, 47, 49, 290, 294

S
scenario, 79, 97, 110, 143, 146, 152, 154, 251, 254, 281
skriptad testning, 56-57, 60-87, 290
skrivare, 124-125
standard, 24, 36, 45, 51, 70, 83, 91, 95, 98, 149, 211, 222, 293, 296
statisk test, 36, 46, 89-91, 102, 290
strukturell test, 52, 92, 94-95, 98, 193, 290
syntax, 94, 101, 211-215

T
taxonomi, 93, 97, 102, 113, 290
tekniska tester, 45, 46, 91, 96, 120, 220, 290
testanalytiker, 24
testbarhet, 29, 37, 79, 96, 223
testfall, 111-113
testkedja, 243, 249-252, 290
testnivå, 44-47, 51-55, 193, 244, 253, 255, 262-266
testpaket, 209, 243, 252-253, 290
teststrategi, 76, 79, 81, 100, 107, 293
testutförande, 44, 56, 62, 66, 67, 68

tidscykel, 217-218
tillståndsgraf, 73, 99, 101, 161-180, 224,236, 274

U
unified process, 47-49
utforskande testning, 30, 56-60, 61-87, 98, 110, 247, 290
utvecklingsmodell, 25, 41-50

V W
verksamhetsprocess, 25, 92, 94, 97, 137-144, 250, 252-255
verktyg, 24, 26, 38, 52, 55, 71, 75-78, 82, 90, 95-96, 100, 231, 238, 247, 257, 258
White Box-test, 73, 92, 94, 102, 290
Whittaker, 75,102, 225, 287, 293
V-modellen, 43-46, 56
W-modellen, 46-47, 292