Den Lilla Svarta Om Teststrategi

Rikard Edgren
med stöd av Henrik Emilsson

Introduktion

Testningens essens: ta reda på vad som är viktigt, och testa det

Testning handlar om att ta fram kvalitetsrelaterad information, och det kan verka enkelt: först tar man reda på vad som är viktigt, och sen testar man det. Men egentligen är det väldigt svårt: det är väldigt många saker som är viktiga, och det finns väldigt många sätt att testa på, och det är inte lätt att få fatt på den information man behöver.

Vi definierar teststrategi som "*vad du ska testa och hur*" eller "*de idéer som guidar testningen*", vilket är mer konkret och testningsnära än vad som ofta används i andra sammanhang. Det är en pågående process, inte nödvändigtvis ett dokument; och ur ett testperspektiv är besluten om teststrategi de kanske allra viktigaste i hela projektet.

Detta ställer en hel del krav på en bra teststrategi:

- konkret - detaljer hellre än fluff
- realistisk - går att genomföra med "normal" turbulens
- befogad - når de testuppdrag som finns
- mångfacetterad – de flesta system behöver testas på flera olika sätt
- resurseffektiv - utnyttjar de resurser som finns utan att slösa
- granskningsbar - går att förstå, och avgöra om den är bra nog
- förankrad - i ledningen, och i testarna
- ändringsbar - så att man kan hantera det oundvikliga oväntade
- felaktig - utan problem, så är den för vag, eller tog för lång tid att skriva

Vi vill försöka beskriva den outtalade kunskap som testare använder när de tar fram en riktigt bra teststrategi för en unik situation. Men även om vi beskriver många strukturer och tankeverktyg att använda, så kan vi inte på riktigt förmedla hur man går tillväga när man förstår att just det sättet passar bra för att nå just det målet. Du behöver se, öva och diskutera för att träna din förmåga till detta. Det är du själv som måste göra jobbet, och du behöver feedback från personer, men också från resultaten av dina egna beslut.

Vi har själva under många år misslyckats med detta; teststrategierna såg bra ut för det otränade ögat, men bestod mest av plattityder som ”*vi ska testa den nya funktionaliteten efter kraven, och automatisera regressionstester i mån av tid*” eller ”*automatiska acceptanstester kryddat med utforskande tester*”. Vi förklarade inte för uppdragsgivarna vad som skulle hända, där fanns nästan ingen koppling till vad uppdragsgivarna ansåg var viktigt, och det gavs inte chansen till andra att tycka till om förbättringar.

Våra grånade insikter från många olika sammanhang får oss att tro att principerna är generella och applicerbara oavsett vilken sorts mix av agilt och vattenfall du arbetar i. Vi hoppas att läsarna av denna lilla bok ska kunna lära sig att ta fram den information, och dra de slutsatser, som behövs för att få bättre teststrategier, och bättre testning.

Några exempel

Det vore inte dumt med ett exempel!

Exempel brukar göra det lättare att förstå, så boken innehåller många teststrategier (rutor med grå bakgrund) som är baserade på sanna historier. I andra halvan av boken finns det två stora exempel, och vi börjar med några små. För att få läsa dessa så måste du först lova att inte sno något rakt av, varje projekt är unikt och behöver en helt egen teststrategi.

SCRUM-möte (muntligt)

1: “Jag kommer testa den nya funktionen på samma sätt som jag brukar (snällt, men inte för snällt;) Säg till om jag ska hjälpa till med enhetstester!”

2: “För PDF-exporten så tror jag att jag ganska snabbt kan ta fram ett gäng representativa exempel-dokument som kommer kunna användas av oss allihop för att se om vår produkts intressanta PDF-aspekter hanteras.”

3: “Ni vet att jag litar på er, men för denna user story, så vill jag vara med och designa detaljerna i de automatiska acceptanstesterna.”

4: “Jag känner att jag har två dagar fritt att göra systemtester jag inte har hunnit med som avser olika fel-lägen. Så utifrån arkitektur-diagrammet, så ska jag försöka provocera många tänkbara fel, och kombinationer av dem, så att produkten inte dyker på ett otrevligt sätt.”

5: “Jag har bokat upp Johan, skatteavdrags-experten, så vi kommer sitta tillsammans idag och utforska de viktigaste sakerna jag inte förstått än.”

Kommentar

Ibland kan en mycket kort, muntlig strategi räcka för att förmedla vad man ska testa. Ofta vet det övriga teamet väl vad du menar, så det behövs inte speciellt mycket detaljer. Se bara till att du är tillräckligt specifik för att ge chansen för andra att komma med tips och önskemål.

Dessa exempel är teststrategi på ganska detaljerad nivå, med många små strategibeslut hela tiden. Var dock vaksam på risken att man fokuserar för mycket på det lilla; även agila projekt behöver ett större system-tänk.

ROPA

Det viktigaste med ROPA är att hjälpa brandkåren ta bra beslut kring stationer och bemanning; centralt för detta är beräkningen av körtider och täckningsgrad.

Vi kommer ta fram en modell av produkten med hjälp av kravdokument, grafiskt användargränssnitt och manualen, och använda den för grundläggande testning av funktionaliteten.
Eftersom det inte finns någon support för ROPA är det viktigt att granska dokumentationen och

säkerställa att felhantering och information i programmet hjälper användarna.
För att testa ROPA mer verklighetstroget kommer vi använda komplexa scenarion som även undersöker pålitlighet och användbarhet.
Som komplettering kommer riskbaserad testning utföras gentemot sekretess-aspekter, installation, och dataintegritet (titta noga på databasens innehåll, stickprov och visuellt.)
Då produkten tidigare inte testats strukturerat, så är en viktig leverabel en lista på buggar i produkten (vi börjar med regressionstestning av tidigare kända fel.)
För att underlätta framtida testning så ska testarna ge riktlinjer för testbarhetsförbättringar; programmatiska gränssnitt som möjliggör automatisk regressionstestning av uträkningar.

Utmaning: I dagsläget finns inget perfekt orakel (rimlighet, Google Maps) för att bedöma om körtidsberäkningarna är korrekta.

Kommentar

Teststrategin för ROPA inleds med en mening som beskriver det som anses viktigast. Om man har fel i sin bedömning, så är det ju mycket bättre att ha det i början än att det inte upptäcks alls! Därefter följer en blandning av informationskällor, tillvägagångssätt och fokusområden. På ett kortfattat sätt beskrivs huvuddragen tillsammans med de risker och kvalitetsegenskaper som bedömdes viktigast. Strategin var lätt att förstå för en icke-teknisk produktägare, och om sanningen ska fram så är ovanstående version inte den första, den innehöll nämligen några missar.

Den stora bristen i strategin är att den inte förmedlar vad man kommer att lägga mest tid på; de besluten är intressanta och viktiga. Ibland kan detta vara väldigt svårt att bedöma, men ett tydligare fokus på vad som prioriteras borde varit med i teststrategin.

Akilles

På nästa sida finns en mind map med den outtalade teststrategin, för en av testarna.

Kommentar

Många bra idéer om hur det ska testas. Bra att namnge tänkta testverktyg; det ger insyn och möjliggör förbättringsförslag. Mer gjord för testare än intressenter (vilket är helt OK för sammanhanget.) Lite väl mycket fokus på kvalitetsegenskaper, känslan är att någon annan typ av utgångspunkt saknas.

Testuppdraget har uppfattats som att testa den nya webbsajten för att identifiera viktiga problem som man vill åtgärda innan, eller strax efter release.

Akilles

Kommer INTE testas
- Datavalidering (enligt överenskommelse)
- Uppdateringar av data
- Installation och uppgradering av servern
- I18n (internationalization - översättningsbarhet)

Kompatibilitet
- IE8 IE9 (Windows)
- Firefox (MAC, senaste)
- Chrome (senaste)
- Safari (iPhone)
- Opera (Android)
- Stickprov på äldre webbläsare

IT-vänlighet
- Undersök om uppdateringar av data kommer vara smidigt

Prestanda
- Testa komplexa scenarier och utvärdera upplevd prestanda
- Använd JMeter för att simulera belastning strax över förväntad
  - fokus på beräkningar
- Ramp-up med JMeter tills det smäller?
- Testa uthålligheten på servern med JMeter-skript som får gå över helgen

Säkerhet
- Kör Zed Attack Proxy
- Kolla versioner på server-programvara (gäller sajten som ska bli publik)

Förmågor
- Testa att publika funktioner gör det de ska: Sök, Visa, Jämför, Beräkna
- Testa att de vanliga funktionerna beter sig som användare rimligen förväntar sig
- Testa publika funktioner efter vanliga användarmisstag
- Testa REST API
- Stickprov på adminhantering av data

Beräkningar
- Felaktiga beräknnigar skadar företagets trovärdighet
- Undersök befintliga enhetstester
- egen, datadriven jämförelseberäkning i Excel
- automatisera Excel och webbsajt med Watir?
- 1 Detta ska vi lägga mest tid på

Pålitlighet
- Korrekthet: Gör även manuella stickprov
- Datavalidering struket ur testuppdraget, men titta lite ändå
- Dataintegritet: Samma data via alla gränssnitt

Användbarhet
- Heuristisk användbarhetsutvärdering
- Låt en ny användare genomföra en jämförelse de är intresserade av
- Kör tillgänglighets-testverktyg för webben (inloggning blir ett problem för många)

Karisma
- Fråga bekanta vad de tycker om sajten
- Ha organisationens grafiska profil i bakhuvudet under tiden som manuell testning görs

Testuppdrag

Att ta fram ett testuppdrag är en garant för att du, dina uppdragsgivare, och omgivning valt att fokusera testningen på rätt saker.

Testuppdraget är svaren på frågan ”**Varför testar vi?**”
Syftet med testuppdrag, från uppdragsgivarnas sida, är att styra testinsatsen. Syftet med testuppdrag från testarnas sida är att få uppdragsgivaren att berätta vad de vill ha information om.

Det är inte ovanligt att testinsatser startas med det alltför vaga uppdraget "ni ska testa produkten". Då är risken naturligtvis stor att testarna inte fokuserar på det viktigaste, och att saker upptäcks för sent eller inte alls.

Generella testuppdrag

Det finns tre överdrivet generella kategorier av testuppdrag: “*hitta problem*”, “*utvärdera produkten*”, “*för att vi måste*”, och ofta är alla tre relevanta. Cem Kaner och James Bach har tagit fram dussinet generella testuppdrag som tjänar väl som inspiration.

* Hitta viktiga fel
* Hitta så många fel som möjligt
* Bedöm kvalitetsrisker
* Certifiera/Kontrollera gentemot standard
* Genomföra/kontrollera regler kring processer
* Gör intressenterna nöjda
* Säkerställa att testningen är redovisningsbar
* Ge råd om kvalitetssäkring
* Ge råd om testning
* Ge råd om kvalitet
* Maximera effektivitet
* Minimera kostnad
* Minimera tid

Dessa behöver förankras i ditt sammanhang och göras mycket mer specifika, det blir exempelvis stor skillnad om minimera tid handlar om utvecklarnas, testarnas eller supportavdelningens tid, det är säkerligen specifika delar där man vill ha råd om testning och kvalitetssäkring.

Dina uppdrag

Du har troligen varianter på flera av dessa som dina outtalade testuppdrag, men du kan också mycket väl ha mer detaljerade uppdrag (som närmast liknar testidéer), till exempel:

* kolla om webbplatsen kommer klara förväntad belastning till julhelgen
* undersök säkerhetsrisker för inloggning och användarkonton

Att hitta rätt detaljnivå är en av de svårare sakerna med formulering av både testuppdrag och teststrategi; är det för vagt så betyder det inte mycket, är det för detaljerat så kan det bli för mycket att överblicka, och detaljstyrt så att den viktigare helheten missas. För testuppdrag rekommenderas att försöka ha dem mer generella, att det beskrivs vilken sorts information man ska leverera, och

vem som behöver den.

Ofta upptäcks ytterligare uppdrag under resans gång, ett eget exempel är att det fanns sekretess-aspekter som inte tidigare hade testats så ordentligt som förväntats. Gå tillbaks till ditt testuppdrag för att se att du är på rätt spår, kommunicera och uppdatera kontinuerligt dina uppdrag; en plan som inte kan förändras är inte bra.

Testuppdrag består oftast av informationsmål, vilket innebär den information som intressenterna vill ha (eller behöver.) De kan ses som en delmängd av projektets mål, och guidar testningen så att rätt testning görs, och i rätt tid. Informationen som framkommer under testning ska lagras och rapporteras i rätt format; och i rätt tid. Om man når sina informationsmål, så lär man ha varit värdefull.

"så att"-knepet

När man kommunicerar fluffiga saker, så är det lätt att tro att man förstår när man kanske inte alls förstår helheten egentligen. ”Så att”-knepet handlar om att tvinga fram förtydliganden, från andra eller från dig själv.

Så när du har ett vagt testuppdrag, till exempel:
testa produkten
lägg till "så att" och fyll på:
så att vi hittar viktiga fel
kanske ytterligare en gång:
så att de kan åtgärdas och ge nöjdare kunder och färre supportärenden
Då börjar du få ett uppdrag som verkligen är meningsfullt, och där personer kan fylla i med mer saker:
så att vi kan ta välinformerade beslut,
så att produktrisker har utforskats, så att vi inte får obehagliga överraskningar
...och det är väl klart att det blir en bättre testning om man har klart för sig vad informationen ska användas till!

Hitta gärna på dina egna knep, t.ex. genom att locka fram detaljer med lockelser som " även om", "såvida inte", “till exempel” o.s.v.)

Undersök nyckelbegrepp

Ditt testuppdrag innehåller troligen nyckelbegrepp som du behöver ta reda på vad de betyder:

"viktiga fel"
"risker"
"kvalitet"

Tro inte att dessa begrepp är lätta att förstå, de behöver utforskas, kanske inte dokumenteras i detalj, men du måste veta mer om dem för att göra en bra testinsats. I en del sammanhang bör varje krasch undvikas, men i andra kanske millimeter-precision på utseendet är viktigare. Om en viktig risk är ”*att användarna kan lita på produkten*”, så behöver du ta reda på detaljer kring vad detta kan innebära, för att sedan kunna designa tester som utmanar produkten i kring området.

Du behöver ställa bra frågor, kanske till många personer, för att ta reda på mer detaljer kring viktiga fel, risker, kvalitet, eller vilket begrepp som nu är återkommande. Om du gjort lite eget förarbete, så

kommer diskussionen ge bättre resultat, och i slutändan kan detta leda till förtydligad förståelse i form av:

> exempel: patchar, klagomål, dåliga recensioner, pinsamheter, buggar.
> riktlinjer: kvalitetsmål, felkatalog, krav, case studies.

Om Jerry Weinberg har rätt i att “kvalitet är värde för en person”, så är det viktigt att ta reda på vilka dessa personer är, och vad de önskar.

Återkoppla till intressenterna

När du kommit en bit på vägen i din förståelse av ditt testuppdrag, så behöver du återkoppla till uppdragsgivarna och andra intressenter. Förklara din tolkning, gärna för många, så att du tolkat även motstridiga förväntningar på ett fruktbart sätt. Du kanske behöver fråga VD:n: "tror du att det här kommer leda till färre obehagliga supportärenden?" Det är inte en nackdel om de upplever det som att det är deras bild du satt ord på.

En mycket viktig insikt som vi har fått genom åren är att intressenterna har sin unika bild av vad som är viktigt i projektet. De tänker väldigt sällan på hur testningen ska genomföras; och i ärlighetens namn har de kanske viktigare saker att tänka på. Det som vi som testare behöver göra är att ta reda på vad det är som de tycker är viktigt, för att sedan se om det de tycker är viktigt är något som testningen kan bidra med att ge information om (allt är inte vårt jobb...)

> Exempel: I ett stort projekt som vi har arbetat i, lyckades vi ta reda på att den högste produktchefen hade som mål att produkten skulle ta sig igenom en upphandlingsscreening. Det vill säga, för att få gå vidare i upphandlingen krävdes att produkten skulle uppfylla 10 viktiga högnivåkriterier för att få gå vidare som potentiell kandidat. Det är en skillnad på att ha som mål att det inte ska finnas allvarliga fel och att tycka att en produkt ska uppfylla ett antal funktionskriterier.
> När vi fick reda på det, kunde vi ha med detta som ett mål för projektet samt designa ett testuppdrag för detta mål: “Undersöka om produkten motsvarar de s.k. 10 kriterierna”. På detta sätt fick vi även idéer om hur vi skulle rapportera resultat från detta testuppdrag.

Outtalade testuppdrag

Det är inte säkert att det uppdragsgivarna vill ha är det de egentligen **behöver**. Alla testinsatser måste inte gå att koppla till ett uppdrag, det är ofta nyttigt att göra stickprov på andra sätt, så att man kan få reda på saker man inte visste var viktiga.

Du har troligen flera uttalade och outtalade uppdrag, om du tar reda på vilka det är, så kan du göra bättre testning och rapportering. Och nöj dig inte med dessa exempel, samarbeta fram specifika uppdrag som gäller för ditt unika sammanhang. Se det gärna som att du ska hjälpa din uppdragsgivare att förankra testuppdraget hos dig.

Testning är ju aldrig bättre än kommunikationen av resultaten, så fundera/diskutera/förankra: **vilken information vill man ha ut av testningen?** Skriv genast ner vilka du ska prata med för att ta reda på vad som egentligen gäller!

Exempel

Här är exempel på dokumenterade uppdrag från en produkt med många olika sorters användare:

* hitta problem som vi vill fixa innan release
* ta fram beslutsunderlag för ledningen

Men där fanns också outtalade uppdrag:

* ge feedback snabbt till utvecklare och kravställare, så att de kan komma fortare fram
* uppmärksamma billiga förbättringsförslag
* verifiera att produkten följer utlovad tillgänglighets-standard
* informera projektledaren om riskers uppgång och fall

Dessa uppdrag var olika viktiga vid olika tillfällen. Det första uppdraget: “hitta viktiga fel” var dock det viktigaste, och det brukar det vara i de flesta sammanhang.

Kommunikation

Testningen är aldrig bättre än kommunikationen av resultaten

Redan från början behöver du fundera på vad och hur du ska rapportera, och vara medveten om att olika intressenter behöver olika information:

* Utvecklare vill ha tydliga buggrapporter och kanske muntlig information om annat
* Testare vill veta vad som har testats
* (Projekt)ledare vill ha status på både produkten och testningens fortskridande

Olika personer tar bäst emot information på olika sätt:

* Några vill ha mail, eller dokument
* Andra vill prata om det
* En del vill veta allt, andra vill att det ska gå snabbt
* För olika personer kan det behövas rapportering varje dag, eller varannan vecka.

Försök prata om hur kommunikationen ska göras, ofta kan man då reda ut missförstånd, eller förstå att man missat något viktigt (kan gälla både testuppdrag och teststrategi.) Om ni är överens om vilken täckning som är viktig, och hur den ska rapporteras, så kommer den framtida kommunikationen bli oerhört mycket lättare. Den som kommunicerar bär ansvar för att informationen tagits emot, så hitta de bästa vägarna, det är inte OK att säga “det skrev vi i testrapporten, men det var ingen som läste.”

Utifrån testuppdragen så vet du vilken sorts information som behövs. Utifrån kommande avsnitt, framför allt kvalitetsegenskaper & risker, så kan du hitta innehållet, och de rätta orden.

Siffror kontra ord

Det kan vara värt att lägga extra krut på att ta reda på vilken information som ska vara kvantitativ (siffror) och vilken som ska vara kvalitativ (ord). Vår erfarenhet säger att bara siffror inte ger en rättvisande bild, och tyvärr är det ofta så att om det inte finns siffror i rapporten, så betraktas den som mindre trovärdig. Samtidigt är det **konversationerna** som är det som påverkar mest, se till att du har många sådana om både strategi och resultat, och att du verkligen förmedlar resultaten på ett ärligt och rättvist sätt.

Det finns så gott som alltid viktiga saker som inte kan mätas med siffror; med vilka ord kan du berätta om dessa? Är din teststrategi inriktad på information i form av siffror eller ord?

Detaljnivåer

I en del sammanhang vill man bara ha resultaten från testningen, exempelvis en lista på viktiga buggar, med den information som krävs för att de ska förstås och kunna fixas. I andra sammanhang behöver man berätta om så gott som alla beslut skriftligt, med referenser till externa källor som bekräftar att fler tycker att man ska göra så. Din verklighet är förmodligen mittemellan, intressenter behöver veta både om resultat och tillvägagångssätt; ibland mycket, ibland mindre, känn efter! “*Något mer du vill veta?*” är en bra fråga, som inte ger alla svar.

Vårt största misstag

Vid ett tillfälle fick vi chansen att testa några tilläggsprodukter som gjordes för ett fåtal kunder. Vi var jätteglada att få hjälpa till och gjorde en snabb och effektiv testning och identifierade dussintals buggar som mailades till utvecklarna. Men vi fick inget svar. De verkade fullt upptagna med annat, och efter ett tag förstod vi att vi totalt missuppfattat vad testuppdraget var. Vi trodde att vi skulle hjälpa genom att hitta så många buggar som möjligt, men det de ville ha var en bekräftelse på att huvud-funktionaliteten var OK, och en allmän bedömning.

Ett kortare samtal om förväntningarna hade löst detta, då hade vi rapporterat de två allvarliga buggar vi hittade. Och värst av allt, vi förstörde ett samarbete som det tog mycket lång tid att reparera.

Smidig rapportering grundläggs genom tydlig kommunikation kring testuppdrag och teststrategi.

Informationskällor

Om det är lätt att ta fram en teststrategi, så vet du nog för lite

Man behöver veta en hel del för att kunna ta fram en användbar teststrategi. Framför allt behöver man veta vad som är viktigt; lättare sagt än gjort...

Det mesta av informationen kommer du få från projektet, och ofta är krav i någon form den bästa starten. Men vill du åstadkomma mer än en uddlös, bekräftande testning, så bör du ta mycket annat i beaktande också. På de kommande två sidorna listas 37 generella informationskällor, som vi rekommenderar att du överväger för fördjupning. Du kan tillsammans med andra, bedöma var man ska dyka djupare för att få rätt inspiration till testningen. Ju mer erfarenhet du har, desto lättare blir det att förstå vilka som är viktiga, och vilka man kan strunta i (tills vidare...)

Du behöver inte fastna i detta i början, likt det mesta i denna bok kan fördjupningar göras kontinuerligt, när det behövs. Ha listan snarare som ett hjälpmedel för att kategorisera de sorters information som är viktigast i din unika situation. På ett företag behölls kategorierna, men allt innehåll gjordes specifikt, och man fick ett bra hjälpmedel för både nya och gamla testare.

Listan siktar mot testidéer, så du behöver bli specifik och även lägga till en dos av abstrakt tänkande för att få material till din teststrategi. Och du kommer också hitta dina orakel (sätt att avgöra om det är ett problem eller inte), kanske behöver de nämnas i din teststrategi?

Prata med intressenter

I startfasen så är det vissa saker som är mer relevanta än andra. Vad dina uppdragsgivare tycker ska beaktas extra noga. För dessa, och övriga, intressenter så är det första rådet: **Låt dem prata**. De vet mycket om vad som är viktigt, och deras egna ord beskriver det bäst. En följdfråga här och där är inte fel, och öppna frågor kan också tas till vid behov:

Något mer som är viktigt?
Vad värderar kunderna?
Kan du ge exempel på viktiga problem som har hänt?
Vad är du rädd ska hända?
Vad borde vi INTE fokusera för mycket på?
Hur skulle du ha velat testa?

Och det handlar inte bara om att lyssna och ställa frågor, man måste förstå svaren, tolka dem i ljuset av vad som ska testas. Viktiga slogans kan ofta tolkas på flera sätt, så se till att du tänkt på många perspektiv.

Exempel: ”*snabbt in, snabbt ut*”, sade en produktägare.
Delstrategi 1: Användartestning för att typiska användare snabbt hittar rätt.
Delstrategi 2: Heuristisk användbarhetsutvärdering med fokus på snabbanvänd (fokus, få klick, snabbläst)
Delstrategi 3: Utvärdera upplevd prestanda då systemet är under normalhög belastning.
Delstrategi 4: Kvalitetssäkra redaktionella processen för att uppdatera vanligaste informationen.

37 källor till testidéer

Vi rekommenderar att du kontinuerligt samlar testidéer från många olika informationskällor.
Titta igenom nedanstående och tänk på risker och värdet av testningen; hitta genvägar för att täcka det som är viktigt.

Produkt	**1. Förmågor.** De mest uppenbara testidéerna handlar om vad produkten förväntas göra. En bra start är krav, exempel och andra specifikationer, eller en funktionslista skapad utifrån programmet. Försök också identifiera outtalade krav, saker som användarna förväntar sig, men som inte dokumenterats. Var beredd på oönskade förmågor. **2. Fel-lägen.** Programvara kan fallera på många sätt, så ställ "tänk om"-frågor för att ta fram tester som undersöker hanteringen av interna/externa, förväntade/oväntade, (o)avsiktliga, realistiska/provocerade fel. Ifrågasätt systemets feltolerans; alla objekt och komponenter kan gå sönder. **3. Modeller.** En tillståndsgraf hjälper dig identifiera testidéer kring tillstånd, övergångar och vägar. En karta över systemets anatomi visar vad som kan testas, och vad som interagerar. Skapa din egen anpassade modell med strukturer som SFDPOT från Heuristic Test Strategy Model. En visuell modell är lättare att kommunicera. Modelleringsaktiviteten brukar ge större förståelse och nya idéer. Många och rika modeller ger bättre testidéer. **4. Data.** När du identifierar vanlig och ovanlig data (det finns alltid brus) så får du en bra start för ett gäng testidéer. Följ enkel och knepig data genom systemet, var inom och utanför gränserna, utmana datatyperna, använd CRUD (Create, Read, Update, Delete), utnyttja beroenden, och titta på datan på många ställen. **5. Omgivning.** Ingen produkt är en ö, så kompatibilitet (hårdvara, OS, program, inställningar, språk) är ett av många viktiga testningsproblem, och undersök också aktiviteter man gör i anslutning till produkten. Om du förstår det stora systemet, så kan du få trovärdiga testidéer, som är svåra att se om man tittar på en funktion i taget. **6. White-box.** En kombination av testarens kritiska tankesätt och utvecklarnas perspektiv utmanar antaganden, och kan hitta fel som är snabba att åtgärda. Titta extra noga på kodvägar och alternativ som är svåra att se ur ett systemperspektiv. Och kodtäckning är inte värdelöst, det kan användas för att hitta saker som inte testats. **7. Produkthistoria.** Gamla problem uppträder gärna i ny skepnad. Sök i ditt bugg- eller supportsystem, skapa en felkatalog, minns kritiska problem och deras rotorsaker. Använd gamla versioner av programvaran som inspiration och orakel. **8. Rykten.** Det finns ofta rykten om produktens kvalitet och problem, varav en del skadar produkten och företaget. Använd ryktena som testidéer; det är ditt uppdrag att ta reda på om de stämmer eller inte. **9. Programmet-i-sig.** När du använder produkten så kommer du få massor av idéer kring vad som är felbenäget, relaterat, intressant. Om du kan äta din egen hundmat (eufemism: sippa din egen champagne), så är det lättare att avgöra vad som är viktigt. **10. Teknologier.** Om du känner till detaljer kring teknologierna som produkten använder sig av, så förstår du vilka områden som är problematiska, du förstår möjligheter och säkerhetsaspekter, du vet vilka parametrar som ändras, och när. Du kan göra rätt variationer, och ha fruktbara, tekniska diskussioner med utvecklarna. **11. Konkurrenter.** Om du tittar på andra lösningar på liknande problem, så kan du se vad som behöver testas, men också få en känsla för vilka egenskaper som slutanvändarna är intresserade av. Det kan finnas interna lösningar (ex. Excel-applikation) att inspireras av, och ofta finns det analoga lösningar för samma syften. Kanske kan du få en briljant idé genom att titta på konkurrenternas support, FAQ med mera?
Bransch	**12. Syfte.** Produktens syften ger dig mål för dina testidéer. Fråga "varför?" några gånger till för att ta reda på de egentliga målen. Dessa kan ge testningen en bred, välvillig start som hittar väldigt viktiga problem, fort. **13. Affärsmål.** Vilka är de viktigaste målen för företaget (och delmål för avdelningar)? Finns det några krav som bryter mot de målen? Känner du till det stora sammanhanget, vet du vad produktens vision och värde är? **14. Produktens image.** Produktens önskade beteende och egenskaper kan vara uttalade eller outtalade, sittandes i väggarna eller i producenternas och konsumenternas tankar. Du kan skriva motiverande buggrapporter om du kan peka på hot mot produktens image, t.ex. genom att visa på avvikelser gentemot marknadsföringsmaterial. **15. Branschkunskap.** Om du känner till sammanhanget produkten befinner sig i, så kan du förstå hur det ger värde till kunderna. Om du inte kan skaffa den kunskapen, samarbeta med någon som har god kunskap om behoven, processerna och miljön. **16. Lagliga aspekter.** Behöver du bekymra dig om kontrakt, böter eller andra legala aspekter? Vad skulle kunna kosta företaget väldigt mycket pengar? Finns det en advokat som kan ge tips om vad som till varje pris ska undvikas?
Team	**17. Kreativa idéer.** Alla produkter är unika och kräver helt nya testidéer. Prova lateralt tänkande (t. ex. Edward De Bonos Sex Tänkarhattar, provocerande operatorer, tvärtom-metoden, slumpmässig stimulans, Google Goggles) för att få kreativa tankar. Metaforer och analogier är ett bra sätt att tänka i nya banor. **18. Interna samlingar.** Använd eller skapa listor över saker som ofta är viktiga i ditt sammanhang. En del kallar detta kvalitetsmönster, andra har produkt-specifika snabbtester. **19. Du.** Du är en användare. Du kan vara en intressent. Du är viktig. Utnyttja dina styrkor från erfarenhet, färdigheter, kunskap och problemkännedom. Använd dina känslor och din subjektivitet för att förstå vad som är viktigt. Om "Kvalitet är värde för en person ", så är det rätt bra om den personen är "jag". Men glöm inte bort dina svagheter och blinda fläckar.

Projekt	**20. Projektets bakgrund.** Orsakerna bakom projektet driver många beslut, och historien bakom tidigare (liknande) projekt är bra att känna till för att kunna göra effektiv testning. **21. Informationsmål.** Det är grundläggande att förstå de uttalade och outtalade målen med testningen. Om du inte har några, skapa egna kvalitets- och informationsmål som kan guida din testning. **22. Projektrisker.** Projektets svårigheter kan delvis hanteras med testning. Du vill veta vilken funktionalitet som utvecklarna har problem med, och du kan ändra din plan beroende på vilka risker som behöver adresseras först. **23. Testartefakter.** Använd dina testidéer, loggar och resultat för fortsatt testning, men titta också på resultat från andra projekt, beta-rapporter, användbarhetsutvärderingar, testresultat från tredjepartskomponenter. Vilka frågor vill du kunna besvara i dina statusrapporter? **24. Skuld.** Genvägar som tas skapar ofta en ökande skuld. Det kan vara projektskuld, lednings-skuld, teknisk skuld, programvaru-skuld, testningsskuld eller vad du vill kalla det. Om teamet har koll på listan över skulder, så kan du skapa testidéer utifrån dem. **25. Konversationer.** Samtal kan ge dig viktigare information än det som finns i specifikationerna. Det finns många som kan hjälpa dig med din testdesign, en del kan bättre bedöma viktighetsgrad, vad kan du få ut av att nämna saker i förbigående? Om utvecklarna litar på att du kan hitta intressanta saker, så kommer de ge dig insider-information om tvivelaktiga delar av programmet. Fråga utvecklarna *"vad tycker du vi borde testa?"* och "*vilken del av koden hade du velat gjort bättre?*" **26. Kontextanalys.** Vad mer i sammanhanget borde påverka vad du testar, och hur? Känner du till marknadsvillkoren och projektets drivkrafter? Är det något som ändrats som leder till nya sätt att testa? Vad testas av andra? Vilka styrkor och svagheter har projektet och dess medlemmar? **27. Många leverabler.** Det finns många saker att testa; det körbara programmet, installatören, API, tillägg, kod och kommentarer, filegenskaper, Hjälp, annan dokumentation, Release Notes, readme:s, reklam, träningsmaterial, demos etc. Alla dessa kan också ha information som kan inspirera. **28. Verktyg.** Om något kan göras väldigt fort är det ofta en bra idé att prova. Verktyg är inte bara ett medel, de kan också vara en startpunkt för utforskning.
Intressenter	**29. Kvalitetsegenskaper.** Det finns alltid kvalitetsegenskaper som avgör om projektet lyckas; ibland är de lätta att nå, ibland kritiska och svåra. Vår definition inkluderar förmågor, pålitlighet, användbarhet, karisma, säkerhet, prestanda, IT-vänlig, kompatibilitet, support, testbarhet, underhåll, flyttbarhet, och en mängd underkategorier. Många av dessa kan användas som pågående testidéer att ha i bakhuvudet; gratis, men redo att identifiera överträdelser. **30. Produkträdslor.** Saker som intressenter är riktigt oroliga för är mycket starkare än risker; de behöver inte prioriteras, de behöver testas. Typiska rädslor som är svåra att verifiera, men värdefulla för testning är: dålig image, fel beslut, skada, ingen gillar produkten. Olika människor har olika rädslor; ta reda på vilka som är viktigast. **31. Scenarion.** Användarna vill åstadkomma eller uppleva något med programvaran, så skapa tester som på ett varierat sätt simulerar sekvenser av produktens beteenden (testa inte bara funktionerna isolerat.) Ju mer trovärdiga användningsmönster du känner till, desto mer realistiska tester kan du utföra. Prova också excentriska såpoperatester för att bredda testtäckningen. **32. Information från fältet.** Förutom kunskap om kundernas problem, miljö, behov och känslor kan du ta dig tid att förstå hur kunderna både lyckas och misslyckas. Intervjua kunder, säljare, marknadsförare, konsulter, support, eller ännu hellre: jobba där ett tag. **33. Användare.** Tänk på olika sorters användare (personer du känner, personas), olika behov, olika känslor och olika situationer. Ta reda på vad de gillar och ogillar; vad de gör i anslutning till din programvara. Du kan spela upp en scen i ditt testlabb och ha rollspel med olika användartyper, vad ger det för testidéer? Bäst är såklart ofiltrerad information direkt från slutanvändarna, i deras sammanhang. Kom ihåg att två liknande användare kan tänka väldigt olika om samma saker.
Externt	**34. Publika samlingar.** Dra nytta av generella eller specifika listor på buggar, kodningsmisstag, eller testidéer. När du bygger upp din egen minneslista, ta en titt på de här: • Appendix A of Testing Computer Software (Kaner, Falk, and Nguyen) • Boris Beizer Taxonomy (Otto Vinter) • Shopping Cart Taxonomy (Giri Vijayaraghavan) • Testing Heuristics Cheat Sheet (Elisabeth Hendrickson) • You Are Not Done Yet (Michael Hunter) Lär dig några tricks och tekniker från böcker, bloggar, konferenser, sök efter tumregler för testdesign, eller uppfinn de som är bäst för dig. **35. Standarder.** Leta upp relevanta bransch-standards, lagar och reglementen. Läs och förstå UI-standards, säkerhetsregler, policy. Kan du hitta information som pekar på luckor i regelverken? **36. Referenser.** Referensmaterial av olika slag är en bra källa till orakel och testinspiration, t.ex. en atlas för en geografisk produkt. All sorts kunskap kan vara användbar, och Wikipedia kan räcka för att få en snabb förståelse av exempelvis en statistisk modell. **37. Sökning.** Att googla är ett bra sätt att hitta saker du letar efter, och saker du inte visste att du behövde (serendipitet.)

This work is licensed under the Creative Commons Attribution-No Derivative License
inspired by Sabourin's 10 Sources for Testing Ideas, Bach/Bolton's HICCUPPS(F) and Kaner's writings.

Produktanalys

Testarens viktigaste färdighet kan vara att lära sig saker snabbt

Hur man ska testa kan inte avgöras utan att veta **vad** som ska testas.
Det finns många sätt att ta reda på detta:

* Börja med kraven
* Brainstorma en lista
* Modellera enligt Bachs SFDIPOT (se Heurisitc Test Strategy Model)
* Resultaten från 37 informationskällor

Vilket som är bäst går inte att säga, men en SFDIPOT är såpass snabb och allmänt användbar att vi gör en fördjupning i denna. Poängen är att få en bild av produkten utifrån många perspektiv.

Struktur – det som utgör själva produkten
Funktioner – det som produkten gör
Data – det som produkten använder
Interaktion – de gränssnitt produkten interagerar med
Plattform – det som produkten är beroende av
Om användande – hur produkten kommer att användas och upplevas
Tid – relationer mellan produkten och tid

Här är några exempel på vad en högnivå-SFDIPOT på ett webb-baserat CRM-system kan ge:

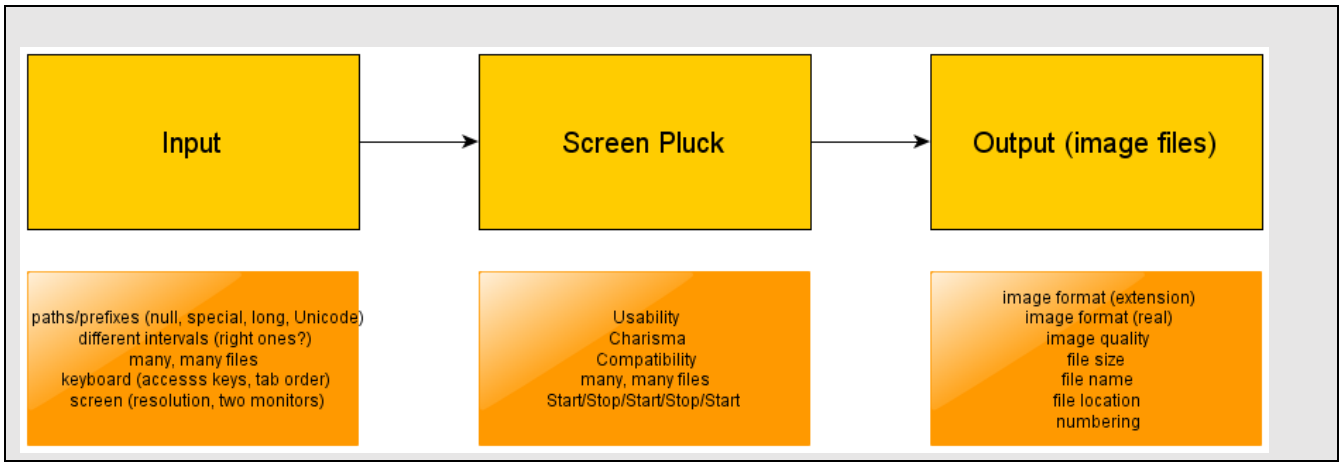
* Struktur gör det tydligt att både en server och en klient behöver testas; prestanda, säkerhet?
* Funktioner ger en överblick över vad produkten innehåller.
* Data kan ge specifika saker att ha med i en eventuell databas-strategi.
* Interaktion leder vägen till API:t, som troligen behöver testas också.
* Plattform leder tankarna till en strategi för att testa olika webbläsare och operativsystem.
* I Om användande kan konkreta produktrisker och kvalitetsegenskaper hamna.
* Tid kryddar man med, vilket ofta ger nya idéer. t.ex. stabilitet om servern kör länge, riskbaserad testning för transaktioner som sker samtidigt m.m.

En mind map passar bra för att göra detta, tänk bara på att inte gå in i för mycket detalj i detta läge, du vill göra en analys som räcker för att skapa en teststrategi. En fördjupad SFDIPOT, kanske på delfunktionalitet, är också kraftfullt, och kan göras när man gör en mer detaljerad testanalys och testdesign. Resultatet kan vara användbart för andra, men allra viktigast kan analyserandet vara, för det ger dig en fördjupad förståelse. Det är vad du lär dig om systemet som kommer ligga till grund för vad du kan och kommer att testa.

Det är inte viktigt var saker och ting hamnar, slå ihop kategorier om det känns rätt, det viktiga är de tankar man får. Kategorin "**Om användande**" är den största och viktigaste i denna fas, och eftersom den kan inrymma både kvalitetsegenskaper och risker, så kommer ett separat kapitel om detta.

Visuella modeller

Om man inte bara arbetar med ord, så kan förståelsen bli tydligare och lättare att kommunicera. En mind map kanske bara är en annan form av en hierarkisk lista, men varför inte prova att rita ett generellt system?



Allra bäst är troligen dina egna sorters modeller, som passar för just ditt specifika behov. En modell ska vara en förenkling, och ska inte vara komplett, arbeta hellre med flera olika, så att förståelse blir bredare. Se också till att du inte blir förälskad i en modell och fokuserar mer på den än på systemet som ska testas.

Produktelement

I slutändan är en produkt en upplevelse eller en lösning på ett problem. Produkter har många dimensioner, så för att testa på ett bra sätt behöver dessa undersökas. Varje kategori nedan representerar en viktig och unik aspekt av en produkt. Testare som fokuserar på bara ett fåtal riskerar att missa viktiga buggar.

Struktur. *Det som produkten består av.*

- *Kod:* strukturer i koden utgör produkten, från exekverbara filer till specifika rutiner.
- *Hårdvara:* hårdvarukomponenter som är integrerade med produkten.
- *Icke-exekverbara filer:* andra filer än multimedia eller program, t.ex. textfiler, exempeldata, eller hjälpfiler.
- *Bidragande material:* annat än programvara och hårdvara som också är en del av produkten, till exempel pappersdokument, webb-länkar och innehåll, paketering, licensavtal etc.

Funktion. *Det som produkten gör.*

- *Applikation:* funktioner som definierar eller utmärker produkten, eller uppfyller centrala krav.
- *Beräkning:* aritmetiska funktioner eller aktiviteter inbäddade i andra funktioner.
- *Tids-relaterat:* inställningar för time-out; dagliga eller månatliga rapporter; nattkörningar; tidszoner; semestrar; ränteberäkningar; villkor och garantier; tidtagningsfunktioner.
- *Transformationer:* funktioner som ändrar eller transformerar något (ex. typsnitt, infoga bild, ta ut pengar från konto.)
- *Uppstart/Avstängning:* varje metod eller gränssnitt för anropande eller initiering, och dessutom att stänga av produkten.
- *Multimedia:* ljud, bilder, video, eller annan grafisk visning i produkten.
- *Felhantering:* funktioner som upptäcker och reparerar fel, inklusive alla felmeddelanden.
- *Interaktioner:* funktioner inom produkten som påverkar varandra.
- *Testbarhet:* funktioner som hjälper till vid testning av produkten, såsom diagnostisering, loggfiler, asserts, testmenyer etc.

Data. *Det som produkten använder.*

- *Indata:* all data som används av produkten.
- *Utdata:* all data som produkten ger som resultat.
- *Förinställningar:* all data som levereras tillsammans med produkten, eller är inbyggt, som till exempel populerade databaser, default-värden m.m.
- *Persistens:* all data som lagras internt och förväntas existera över flera operationer. Detta inkluderar tillstånd i produkten, såsom inställningar, visningslägen, innehåll i dokument m.m.
- *Sekvenser/Kombinationer:* ordnad eller permuterad data, t.ex. ordningsföljd, sorterad vs. osorterad data, körordning för tester.
- *Stor/Liten:* variationer i storlek och aggregering av data.
- *Brus:* data eller tillstånd som är ogiltigt, korrupt, eller producerat på ett okontrollerat eller felaktigt sätt.
- *Livscykel:* förändringar över dataenhetens livstid; skapa, läsa, ändra, och ta bort.

Interaktion. *Gränssnitt genom vilka produkten ges åtkomst eller uttrycks.*

- *Användargränssnitt (UI)::* element som förmedlar utbyte av data med användaren (e.g. navigering, uppvisning, datainmatning).
- *Systemgränssnitt:* element som utbyter data med annat än användaren, såsom andra program, hårddisk, nätverk, skrivare, API etc.
- *Import/export:* funktioner som paketerar data för användning av annan produkt, eller for att tolka data från annan produkt.

Plattform. *Det som produkten är beroende av (och utanför projektet).*

- *Extern hårdvara:* hårdvarukomponenter och konfigurationer som inte är en del av produkten, men som krävs (eller valbart) för att produkten ska fungera: CPU, minne, tangentbord, kringutrustning etc.
- *Extern programvara:* programvarukomponenter och konfigurationer som inte är en del av produkten, men som krävs (eller kan läggas till) för att produkten ska fungera: operativsystem, samtidiga applikationer, drivrutiner, typsnitt etc.
- *Interna komponenter:* moduler och andra komponenter som används av din produkt, men produceras utanför projektet.

Om användande. *Vem/Var/När/Hur/Varför produkten kommer användas.*

- *Användare:* egenskaperna för olika sorters användare.
- *Miljö:* den fysiska miljö som produkten används i, inklusive saker som oväsen, ljud och distraktioner.
- *Vanlig användning:* mönster och sekvenser av inmatning som produkten vanligen möter. Varierar per användare.
- *Ej uppskattad användning:* mönster från okunnig, felaktig, oförsiktig eller fientlig användning.
- *Extrem användning:* utmana mönster och sekvenser av inmatningar som är förenligt med den avsedda användningen av produkten.

Tid. *Relationer mellan produkten och tid.*

- *Indata/Utdata:* när indata ges, när utdata skapas, och alla tidsrelationer mellan dem (fördröjningar, intervall m.m.)
- *Snabbt/Långsamt:* testa med “snabb” eller långsam inmatning; snabbast och långsammast; kombinationer av långsamt och snabbt.
- *Ändrad hastighet:* snabba upp och sakta ner (spikar, utbrott, hängningar, flaskhalsar, avbrott).
- *Samtidighet:* mer än en sak samtidigt (många-användare, tidsdelning, trådar, och semaforer, delad data).

Av James Bach, översatt av Edgren/Emilsson

Kvalitetsegenskaper & Risker

Allt som kan mätas är inte viktigt, och allt som är viktigt kan inte mätas (Einstein)

Det kan i projektet ha gjorts en riskanalys (som du förhoppningsvis deltagit i) som kommer innehålla två sorters risker:

* projektrisker - singelkompetens, svårprogrammerat m.m.
* produktrisker - nyckelfunktionalitet, prestanda, användbarhet m.m.

Projektriskerna kan testningen hjälpa genom att prioritera och tidigarelägga testning kring dessa områden. Samma sak för produktriskerna, men dessa kan också användas som allmänna riktlinjer, speciellt om de formulerats som kvalitetsegenskaper, eller kvalitetsmål.

Vi rekommenderar att riskerna är en pusselbit (av flera) för din teststrategi, du ska alltså inte bara prioritera tester utifrån dessa. Det är ofta fruktbart att göra riskbaserad testning genom att göra allt man kan för att få en produktrisk att hända, och om det händer, ja då undersöker man hur troligt det är.

På de följande sidorna kommer en lång lista på egenskaper som kan vara relevanta för ditt system; använd dessa som en kraftfull checklista för att utöka, eller förändra, de kvalitetsmål som redan finns (finns inga, så kan testarna själva skapa en, och dela med sig av.) Men kopiera inte text rakt av, utan se till att du förankrar kvalitetsmål i specifika och konkreta saker i ditt system. För mycket fluff kan förvirra, men därmed inte sagt att kvalitetsmål måste vara mätbara.

Hitta gärna på egna ord som passar bättre i sammanhanget, till exempel borde olika aspekter ”spelbarhet” vara kvalitetsbärare för spel. Och lyft gärna upp en underkategori som den viktigaste, exempelvis namngivna beräkningar som behöver testas riktigt noga.

Din teststrategi bör täcka de viktiga kvalitetsegenskaperna, men tänk på att det är en många-till-många-relation; ibland fångar en delstrategi många saker, ibland behövs flera delstrategier för en kvalitetsegenskap. För de allra viktigaste egenskaperna kan man prova att göra “Davids manöver” som innebär att man gör en SFDIPOT-analys för just en viktig kvalitetsegenskap.

Att lista ut vilka kvalitetsaspekter som eftersträvas kan göras fort (man vet mycket sedan förut) eller långsamt (man pratar med många intressenter om vad som är viktigt). Att sedan komma på hur man ska testa dem bygger förstås mycket på erfarenhet och igenkänning av olika sätt att testa.

Strategi-exempel baserat på kvalitetegenskaper:

Stabilitet: utveckla en semi-realistisk robot som kan köra produkten över kvällar och helger...
Dataintegritet: …med slumpmässig data och inbyggd datavalidering.
Robusthet/Stresstålighet: utmana produktens viktiga begränsningar…
Återhämtning: …och undersök hur väl den återhämtar sig efter allvarliga (provocerade) fel

Kvalitetsegenskaper för programvara

Gå igenom listan och tänk på din produkt och dess funktioner. Utgå från din kontext och gör om listan till din egen.

Förmågor. *Erbjuder produkten värdefulla funktioner?*

- *Komplett:* alla viktiga funktioner som önskas av användare är tillgängliga.
- *Korrekt:* utdata eller beräkningar i produkten är korrekta och presenterade med signifikanta siffror.
- *Effektiv:* utför sina funktioner på ett effektivt sätt (utan att göra något den inte ska).
- *Interoperabilitet:* olika funktioner interagerar med varandra på bästa sätt.
- *Samtidighet:* möjlighet att utföra multipla parallella uppgifter, och att kunna köra samtidigt som andra processer.
- *Dataagnostisk:* stödjer allehanda dataformat och hanterar databrus.
- *Utökningsbar:* möjlighet för kunder eller tredjepartsleverantörer att lägga till egna funktioner eller förändra beteendet.

Pålitlig. *Kan du lita på produkten i många och svåra situationer?*

- *Stabilitet:* produkten ska inte krascha , orsaka undantag eller skriptfel.
- *Robust:* produkten hanterar (o)förutsedda fel på ett behagligt sätt.
- *Stresstålighet:* hur beter sig systemet när olika gränser överskrids?
- *Återhämtning:* det är möjligt att starta om och fortsätta efter ett allvarligt fel.
- *Dataintegritet:* all sorts data behålls intakt genom hela produkten.
- *Säkerhet:* produkten medverkar inte till att skada personer eller egendom.
- *Katastrofhantering:* vad händer om någonting riktigt, riktigt allvarligt inträffar?
- *Trovärdighet:* är produktens beteende konsekvent förutsägbart och trovärdigt?

Användbarhet. *Är produkten lätt att använda?*

- *Inbjudan:* produkten inbjuder till att upptäcka möjligheter i produkten.
- *Intuitiv:* det är lätt att förstå och förklara vad produkten kan göra.
- *Minimalistisk:* produktens innehåll eller utseende innehåller inget onödigt.
- *Lättlärd:* det går snabbt och lätt att lära sig hur produkten ska användas.
- *Ihågkombarhet:* har du lärt dig hur du ska utföra något, glömmer du inte bort det.
- *Utforskningsbar:* information och möjligheter om produkten kan upptäckas genom att utforska användargränssnittet.
- *Snabbanvänd:* en erfaren användare kan utföra vanliga uppgifter väldigt snabbt (t.ex. tangentbordsnavigering)
- *Interaktivitet:* produkten har lättförståeliga tillstånd och utmärkta möjligheter till interaktion via användargränssnittet.
- *Kontroll:* användarens känsla av kontroll över programvaran genom hela användarscenariot.
- *Klarhet:* är allting explicit och detaljerat uttryckt, med ett språk som kan bli förstått och inte är tvivelaktigt?
- *Felhantering:* felmeddelanden är informativa, det är svårt att göra misstag och lätt att reparera.
- *Konsekvent:* beteenden är samma genom hela produkten, och det är igenkännligt utseende och känsla.
- *Anpassningsbart:* förvalda inställningar och beteenden kan specificeras för att uppnå flexibilitet.
- *Tillgänglighet:* produkten kan användas även av personer med funktionsnedsättningar, och möter tillgänglighetsriktlinjer.
- *Dokumentation:* det finns en Hjälp som hjälper och som beskriver den aktuella funktionaliteten.

Karisma. *Har produkten "det"?*

- *Unikhet:* produkten är säregen och har någonting som andra inte har.
- *Tillfredsställelse:* hur känner du dig efter att ha använt produkten?
- *Professionalism:* andas produkten rätt typ av professionalism och känns den anpassad för syftet?
- *Attraktivitet:* är alla aspekter av produkten estetiskt tilltalande för ögon och andra sinnen?
- *Väcker nyfikenhet:* kommer användarna bli intresserade att prova sig fram i produkten?
- *Uppslukande:* blir användarna fast, satta i trans, och helt uppslukade när de använder produkten?
- *Hajp*: använder produkten för mycket eller för lite av den bästa och senaste tekniken och idéerna?
- *Över förväntan:* produkten överträffar dina förväntningar och möter de behov du inte visste du hade.
- *Attityd:* har produkten och dess information den rätta attityden och pratar till dig med rätt språk och stil?
- *Förstaintryck:* är (första) intrycken av produkten imponerande?
- *Historier*: finns det intressanta historier om produktens uppkomst, konstruktion eller användande?

Säkerhet. *Stoppar produkten oönskat användande?*

- *Autentisering*: produktens igenkännande av användare.
- *Auktorisering*: produktens hantering av vad autentiserade användare kan se och göra.
- *Privathet*: förmågan att inte avslöja data som är skyddad mot icke-auktoriserade användare.
- *Säkerhetsluckor*: produkten ska varken erbjuda eller inbjuda till sårbarhetsintrång.
- *Hemlighetsfull:* produkten ska under inga omständigheter avslöja information om underliggande system.
- *Osårbarhet:* förmågan att motstå försök att penetrera säkerheten.
- *Virusfri:* produkten ska inte vidarebefordra virus eller ge intrycket av att vara ett.
- *Piratkopieringsskydd:* ingen möjlighet att illegalt kopiera och distribuera programvaran eller koden.
- *Reglementsenlig:* produkten uppfyller fastställda säkerhetsstandarder.

Prestanda. *Är produkten tillräckligt snabb?*

- *Kapacitet:* produktens möjligheter/begränsningar, i många olika situationer (t.ex. slött nätverk)?
- *Resursutnyttjande:* välanpassat användande av processor, minne, lagring och andra resurser.
- *Respons:* (den upplevda) snabbheten när funktioner utförs.
- *Upptid:* systemet är tillgängligt för användande vid alla tidpunkter det ska vara.
- *Genomströmning:* produktens förmåga att processera många saker samtidigt.
- *Uthållighet:* kan produkten hantera last över lång tid?

- *Återkoppling:* svarar systemet tillräckligt snabbt när användare utför funktioner?
- *Skalbarhet*: hur bra skalar systemet upp, ner, eller ut?

IT-vänlig. *Är produkten lätt att installera, underhålla och supporta?*

- *Systemkrav:* förmågan att köra på stödda konfigurationer och att hantera olika miljöer eller saknade komponenter.
- *Installerbarhet:* produkten kan installeras på avsedda plattformar och använder en rimlig storlek av hårddiskutrymmet.
- *Uppgradering:* det är enkelt att uppgradera till en ny version utan att tappa konfigureringsdata och inställningar.
- *Avinstallation:* alla filer (förutom användar- eller systemfiler) och andra resurser ska tas bort under en avinstallation.
- *Konfiguration:* kan installationen konfigureras på olika sätt eller ställen för att stödja kundens användande?
- *Utrullningsbarhet:* produkten kan rullas ut av IT-avdelningen till olika typer av (begränsade) användare och miljöer.
- *Underhåll:* är produkten och dess artefakter lätta att underhålla och supporta för kunderna?
- *Testbarhet:* hur effektivt kan den utrullade produkten bli testad av användarna?

Kompatibilitet. *Hur väl interagerar produkten med andra programvaror och miljöer?*

- *Hårdvarukompatibilitet*: produkten kan användas tillsammans med applicerbara konfigurationer av hårdvarukomponenter.
- *Operativsystemkompatibilitet*: produkten kan köra på avsedda versioner av operativsystemen och följer deras riktlinjer.
- *Programkompatibilitet:* produkten, och dess data, fungerar ihop med andra program som kunderna använder.
- *Konfigurationskompatibilitet*: produktens förmåga att smälta in med omgivningens konfigurationer.
- *Bakåtkompatibilitet:* kan produkten göra allt den förra versionen kunde?
- *Framåtkompatibilitet:* kommer produkten att kunna använda artefakter eller gränssnitt i framtida versioner?
- *Hållbarhet:* påverkan på miljön, t.ex. energieffektivitet, elbesparningslägen, möjlighet till distansarbete.
- *Reglementsenlig:* produkten uppfyller applicerbara standarder, regelverk, lagar, riktlinjer eller etiska förhållningssätt.

Interna kvalitetsegenskaper för programvara

Dessa egenskaper är inte alltid något som upplevs av slutanvändarna, men de kan vara lika viktiga för produkten.

Support. *Kan kundernas användning och problem understödjas?*

- *Identifierbar:* det är lätt att identifiera delar av produkten och deras version, eller specifika fel.
- *Diagnostisering:* är det möjligt att få reda på detaljer kring kundsituationer?
- *Undersökningsbar:* är det lätt att sätta fingret på fel (t.ex. loggfil) och hjälpa?
- *Avlusning:* kan du observera programmets interna tillstånd när det behövs?
- *Mångsidighet:* möjligt att använda produkten på fler sätt än vad det designades för.

Testbarhet. *Är det lätt att verifiera och testa produkten?*

- *Spårbar:* produkten loggar händelser på lämplig nivå, och i användbart format.
- *Styrbar:* möjligheten att sätta tillstånd, objekt och variabler.
- *Observerbar:* möjligheten att observera det som testas.
- *Övervakningsbar:* kan produkten indikera hur och vad den gör?
- *Isolerbar:* möjligheten att testa en del för sig själv.
- *Oföränderlig:* förändringar till programvaran görs kontrollerat, och inte för ofta.
- *Automation:* finns det publika eller interna programmatiska gränssnitt som kan användas?
- *Information:* möjligheten för testare att lära sig det man behöver veta...
- *Granskningsbar:* kan produkten, och sättet den skapats på, valideras?

Underhåll. *Kan produkten underhållas och utökas till låg kostnad?*

- *Flexibel:* är det lätt att ändra produkten för att möta kunders behov?
- *Utökningsbar:* blir det lätt att lägga till funktionalitet i framtiden?
- *Enkel:* koden är inte mer komplex än vad som krävs, och försvårar inte testdesign, testekekvering och resultattolkning.
- *Läsbar:* koden är dokumenterad och lätt att läsa och förstå.
- *Genomskinlig:* Är det lätt att förstå de underliggande strukturerna?
- *Modulär:* koden är uppdelad i lätthanterliga bitar.
- *Omskrivningsbar:* är du nöjd med enhetstesterna?
- *Analyserbar:* möjligheten att hitta orsaker till problem, eller annan intressant kod.

Flyttbarhet. *Är det möjligt att flytta produkten till andra miljöer och språk?*

- *Återanvändning:* kan delar av produkten användas på andra ställen?
- *Anpassningsbar:* är det lätt att förändra produkten så den fungerar i en annan miljö?
- *Kompatibel:* följer produkten vanliga beteenden, eller officiella standards?
- *Internationalisering*: det är lätt att översätta produkten.
- *Lokalisering:* är alla delar av produkten anpassade för att möta behoven av det nya landet/kulturen?
- *Robusta gränssnitt*: kommer produkten se lika bra ut när den översatts?

Rikard Edgren, Henrik Emilsson and Martin Jansson - thetesteye.com v1.1.1

Detta arbete är licensierat under Creative Commons Erkännande -IngaBearbetningar 2.5 Sverige
inspirat av James Bachs CRUSSPIC STMPL, ISO 9126-1, Wikipedia:Ilities och mer...
översatt från http://thetesteye.com/posters/TheTestEye_SoftwareQualityCharacteristics.pdf

Projektomgivning

Man kan inte två gånger kliva ner i samma flod (Herakleitos)

Det är inte bara systemet som byggs som är unikt, sammanhanget du befinner dig i är ännu mer unikt. Du bör dra nytta av möjligheter, och hantera begränsningar i den verklighet testningen ska bedrivas i. Det är här det pragmatiska tänkandet kommer in; vad kan du åstadkomma med det som finns, vilka förutsättningar behöver förbättras?

Nedan följer några kategorier att överväga, men i din situation kan det vara helt andra saker som är avgörande. Det svåra ligger i att förstå vilka delar av projektomgivningen man ska undersöka noga, och vilka man kan ignorera.

Intressenter

Vissa intressenter är uppenbara, och du har förhoppningsvis redan pratat med dem. Det kan också finnas dolda intressenter som sitter inne på viktig information, t.ex. ekonomiavdelningen, advokaten, kunden som vägrar uppgradera, IT-chefen på största kunden m.m.

I en teststrategi tycker vi att man gärna får namnge personer, då blir det större chans att man förstår mer, och verkligen pratar med fler.

Utvecklare

Ofta är det utvecklare man samarbetar mest med, så det är viktigt att förstå hur man kan hjälpa dem på bästa sätt. Det handlar dels om hur man kommunicerar resultaten, men framför allt om vad man väljer att testa. Utvecklare har ofta bra idéer, ta till vara på dem, men testa också andra saker som bedöms viktiga. Om du känner till utvecklarnas starka sidor, så kan du testa mindre där, och fokusera mer på deras svagheter, så länge det är viktiga områden. Berätta om din teststrategi, det har vi själva gjort alltför sent vid flera tillfällen.

Schema

Vilka deadlines och milstolpar påverkas testningen av? Vad händer vid förseningar?
Behöver du bestämma i vilken ordning olika sorters testning ska göras?

> Strategi-exempel: Om ditt testuppdrag innehåller “hjälpa utvecklarna skriva bra kod”, så vill du troligen testa så nära programmerarna som möjligt i både tid och rum.

Förra gången

Om ditt projekt gör en ny version så ska du använda förra vändans strategi på ett förnuftigt sätt. Börja med att slänga allt och tänka fritt; ta sedan fram den gamla och inspireras av de testmetoder som var bra förra gången. Om man gör tvärtom, så är risken stor att strategin blir en blek kopia som inte passar lika bra denna gång.

Testare

Testgruppen är de som kommer utföra merparten av testningen, så man behöver anpassa sin strategi så att den passar verkligheten.

Vilka styrkor och svagheter har de som ska testa? Kan du lita på dem?
Får de göra avvikelser och testa bredare än planerat?
Behöver kompetens höjas eller hyras in?
Finns den mångfald som behövs för att utvärdera produkten ur många olika perspektiv?
Vilka andra vill vara med ibland?

Teststrategin är ett hjälpmedel för att vi ska nå bra resultat, och om testgruppen är oerhört mycket bättre på en viss sorts testning, så kan det vara rätt att testa så, även om en annan metod i teorin skulle varit mer lämplig.

> Strategi-exempel: Vi kommer ha en halvdags workshop där vi går igenom vad man ska titta efter vid tester av autenticering/auktorisering. Vi kommer även att ta upp aspekter som testarna alltid ska ha i bakhuvudet, till exempel att systemet är hemlighetsfullt, det ska inte avslöja detaljer om databas eller annat i felmeddelanden.

Testmiljö

Man testar alltid i en omgivning, och det är sällan testmiljön är så simpel att den handlar om en enskild dator för slutanvändare. Ofta behövs både servrar och klienter, du behöver testdata som påminner om kundernas, du behöver databaser som är korrekt uppsatta (eller med typiska misstag...) Du vill ha en omgivning som gör att du kan lita på resultaten, och en plats där du kan övervaka, vågar utmana systemet och enkelt återställa när det havererar. Ibland måste man testa i produktionsmiljö ändå, beakta då i din strategi att det lär bli snälla, bekräftande tester som utförs.

Vi har en bred bild av vad teststrategi är, så när du obfuskerar verklig data, så tycker vi att du arbetar med din strategi för testmiljön, och det behöver man göra ofta.

> Strategi-exempel: En olöst utmaning är att skapa komplex testdata som motsvarar den som kunderna kommer använda.

Testverktyg

Verktyg kan snabba upp vårt arbete och hjälpa oss att testa saker som annars inte vore möjligt. För vår teststrategi så behöver vi fundera på vilka sorters verktyg som är lämpliga; för svåra problem behöver vi till och med fundera på om det behöver byggas specialverktyg för unika behov. Fundera gärna på produktens testbarhet, som kan möjliggöra användning av testverktyg, och ibland göra att de externa verktygen inte behövs.

Testare är allt som oftast snabblärande, så ovanför verktygserfarenhet tror vi att “förmågan att förstå behov för testverktyg, och dess begränsningar” är viktigast. Detta gäller framför allt området manuellt kontra automatiserat, en ofta missvisande frågeställning som tas upp alltför tidigt. **När man förstår detaljerna brukar det inte vara svårt att välja vilket sätt att testa som passar bäst.**

Brasklapp: Om du använder ett testverktyg mycket, så kommer du nog aldrig kunna byta ut det.

> Strategi-exempel: Prestandatestning ska i möjligaste mån utföras med JMeter, på samma sätt som i förra releasen, så att jämförelserna blir användbara.

Andras testning

Det görs alltid en mängd testning av andra än testare. Det innebär ofta att man kan testa vissa saker mer lättviktigt (men sällan strunta i det helt,) Det kan också finnas outnyttjade möjligheter att flika in lite test-tänk här och var.

Typiska test-aktiviteter att förhålla sig till:

* utvecklare testar att koden gör det den ska
* utvecklare skriver enhetstester/integrationstester
* tekniska skribenter försöker använda produkten när de ska beskriva den
* produktägare m.fl. demonstrerar hur bra programmet är
* slutanvändare provar sig fram
* support försöker hjälpa någon i nöd
* applikations-konsulter bygger en kundanpassad lösning

Oftast vill man ha resultaten innan release, men informationen från ovanstående kan vara nyttig (inspiration) närsomhelst. Riktigt viktiga saker vill man automatiskt kunna fånga om de händer när systemet är live.

> Strategi-exempel: När testare tycker till om användarupplevelsen, så lyssnar ingen, därför kommer vi snarare dubbelkolla att produktägaren tycker till om detta.

Ansvar

I mer formella sammanhang kan den viktigaste delen av en teststrategi vara att visa vem som bär ansvar för olika delar; att tydliggöra vilka avgränsningar och risker som beslutsfattarna skriver under på.

Är du ansvarig för fler delar av kvalitetssäkringen? Kanske är granskningar en viktig del?

Testdesign

Helheten skapar förståelsen för detaljer; detaljerna skapar helheten

Det är nästan ofrånkomligt att man börjar designa tester under tiden man sätter sig in i materialet. Detta är naturligt och bra; enkla och naiva beskrivningar av tester kan mycket väl vara de bästa att starta med. Vi rekommenderar att man kontinuerligt skriver ner dessa idéer, gärna i form av "enradingar" som fångar essensen. (Av tidsskäl gör man inte detta för det mest uppenbara, det som kommer fångas ändå.) Om du gör en lista på viktiga och intressanta testidéer, så kommer gladeligen flera i din omgivning att läsa och ge återkoppling med ändringar och tillägg.

I en teststrategi vill man för det mesta hålla det på en högre nivå. Är man för detaljerad så blir det oläsligt för intressenter, och tar för lång tid att skapa. Här kommer svepande testtekniker väl till pass, metoder för att testa som du tror kommer vara lämpliga. Dessa är de vanligaste generaliserade typerna:

* **Funktionstestning** - man testar att alla funktioner gör det de ska (och inte gör det de inte ska.)
* **Riskbaserad testning** - man testar för att se om de viktigaste riskerna kan inträffa.
* **Specifikationsbaserad testning** - man utgår från påståenden om produkten (inte nödvändigtvis bara specar), och ser om de stämmer.
* **Scenariotestning** - man testar längre sekvenser av händelser, gärna med komplexitet för sekvenser, användare, data och/eller miljö.
* **Modellbaserad testning** - testar utifrån tillstånd, arkitektur, flöden, specialanpassade modeller.
* **Kvalitetsmålsbaserad testning** - Varje kvalitetsegenskap kan ge upphov till en testteknik, e.g. prestanda, säkerhet, användbarhet, kompatibilitet (och underkategorier.)
* **Högvolymstestning** - Kör väldigt många tester för att undersöka stabilitet, använda "all" data, se mönster m.m.
* **Domäntestning** - Välj data utifrån ekvivalensgrupper, gränsvärden eller bästa representanter.
* **Användartestning** - Låt (simulerade) användare utföra uppgifter.
* **Testning utan krusiduller** - Man vet vad som behöver testas, och gör det.

Därtill kommer skriptad (bestämt i förväg) och utforskande testning (testaren vid rodret), som är stilar som kan användas på alla dessa. Vad som är lämpligt att köra manuellt, med verktyg, eller helautomatiserat brukar ofta lösa sig efterhand. Om man ser manuellt och automatiserat som i konflikt, så har man troligen tänkt fel någonstans.

Det är viktiga val som görs här, men inte så viktiga att de inte kan ändras. Ofta använder man sina egna varianter på testtekniker; det viktiga är att guida testningen i rätt riktning, mot testuppdraget. Duktiga testare kommer reparera brister i strategin, eftersom de lär sig mer om systemet, och hur det bäst testas. Försök vara konkret genom att koppla **vad** till **hur**.

> Strategi-exempel: Användbarhetstestning överlåts till fokusgruppen som har den rätta bakgrunden, men individuella testare kommer också ha ett vakande öga på detta.

> Strategi-exempel: Testgruppen kommer använda OWASP ZAP för att se hur säkerheten verkar. Om problem hittas, så har utvecklarna bollen. Om problem inte hittas, bör vi överväga att ta in en pen-testare.

Att lära sig teststrategi

Testning är svårt, tacka gudarna för det!

Det är inte så svårt att komma på många olika sätt att testa programvara för att nå ett uppdrag. Det är däremot mycket svårt att veta vilka vägar som är bra, det är svårt att bestämma vilka varianter som ska undersökas mer noggrant.

Låt oss jämföra med schack, som inte är lika komplext (ett bräde, två spelare, 32 pjäser, tydliga regler) men där ändå varje parti är unikt. En nybörjare har mycket svårt att utvärdera vilka drag som är bra, och kommer göra många misstag. En dator har lärt sig regler för poängsättning av en position, och kan med sin beräkningskapacitet processa många dragkombinationer, och därmed spela riktigt bra. En stormästare i schack räknar inte på speciellt många fler drag än en amatör, däremot så vet stormästaren instinktivt vilka drag som kan vara bra, och vilka som inte är det. Stormästaren har kunskap om olika schackstrategier; stor erfarenhet av olika vägar att ta i olika situationer; och har tränat många gånger på att välja väg i partierna. Stormästaren har förvärvat en tyst kunskap som gör att de på bråkdelen av en sekund kan se bättre drag än en amatör gör efter femton minuters funderande. Förutom att ha sett och upplevt tusentals strategier, har hen en förmåga att ignorera det irrelevanta.

Det kan vara likadant med teststrategi; man behöver träna i många situationer; se och höra om andras val och resultat; man behöver diskutera, vara kreativ, och skapa en förståelse för hur olika teststrategier kan fungera i olika sammanhang. När vi lär ut teststrategi så gör vi gemensamma teststrategier i klassrummet, för vissa saker går inte att läsa sig till. Vi berättar hemligheten att de som kommer att diskutera testning mycket, kommer radikalt accelerera sin förmåga som testare. Mycket av intuitionen kommer från igenkänning, så ju fler teststrategier du tar del av, desto duktigare kommer du bli.

Helheten och detaljer

I samband med att du börjar få klart för dig hur testningen ska gå till, så gör du något mycket komplicerat: du kombinerar förståelsen av helheten med vissa detaljer, och får en mental bild över vad som ska testas och hur. Sådana komplexa tankeaktiviteter tränas bäst genom att just träna (och diskutera.) Dina första idéer kan vara de bästa, men ta hjälp av fler för att säkerställa att du inte missat något viktigt **vad**, eller ett bättre **hur**.

Du syr ihop säcken genom att lägga till kitt, sådant som är viktigt i just ditt sammanhang. Det är därför det inte finns någon bra mall, ramarna för en teststrategi måste variera. I vissa fall kanske teststrategin centrerar kring hur man ska testa relevanta kvalitetsegenskaper, i andra utgår den från funktionsområden eller resurser eller processen runtomkring, i tredje är det testtekniker och verktyg som är det bästa sättet att berätta om strategin, vilket görs muntligt på veckomötet.

Den skrivna teststrategin

Generellt sett, var mer specifik

När du kommit fram till hur testningen bör göras, så behöver man ofta skriva ner detta, dels för tydlig kommunikation och möjlighet till feedback, men också för att man lär sig genom att skriva. Det finns två grupper av läsare

* Testare - som ska få guidning för att göra rätt testning
* Ledningen - som ska lita på att du kommer nå deras mål på ett bra sätt

Testarna kommer du troligen ha många chanser att förtydliga och förbättra strategin med, men ledningen kanske du bara får en chans att kommunicera strategin till. Välj därför orden noga, och se till att de kan förstås även av icke-testare; eller kanske skriva två strategier för olika läsare?

Det viktigaste med teststrategin är innehållet, men hur det formuleras styr huruvida ledningen kommer att

* förstå vilken testning som ska göras
* få förtroende för testgruppen
* kunna ge förslag på förbättringar, exempelvis genom att förtydliga testuppdraget
* snabbare förstå resultaten från testningen

Vad som finns med i strategin varierar, men den kan innehålla dessa saker (men måste inte!)
Du tar bara med saker som är intressanta, som ”**ökar förståelsen**” och ”**möjliggör feedback**”.
Kontrollfrågorna kan hjälpa dig att tänka ett varv till.

Innehåll	**Förklaring**	**Kontrollfrågor**
det uppenbara	det du vet är viktigt, och som behöver täckas av strategin	Fångar din strategi sådant som absolut inte får gå fel? Är det självklara självklart för alla?
testuppdrag	anledningarna till att testning utförs	Är testuppdraget förankrat i ledningen, och i testarna? Finns det verkligen inte fler anledningar att testa?
intressenter	personer som påverkas av produkten och testningen	Vad heter intressenterna? Hur hanteras motsägelsefulla intressen hos dem?
testmetoder	förtydliganden om hur testningen ska gå till	Vad innebär metoderna för er? Kan du motivera era val? Har du övervägt andra sätt?
orakel	sätt som används för att bedöma om man ser problem eller inte	Har du googlat ”FEW HICCUPPS”? Vilka orakel är starkast?
informationskällor	den input vi har till testningen	Omfattar din lista på informationskällor även personer? Behöver några undersökas mer innan ni bestämmer teststrategin?

modeller	förenklingar som visualiserar vad som ska testas, men också begränsningar för vårt tänkande	Har du koll på helheten? Kan du analysera detaljerna? Vilka osynliga, mentala modeller används?
kvalitetsegenskaper	relevanta nyckelord och kvalitetsmål för just detta projekt	Vad betyder kvalitetsegenskaperna i er unika situation? Kan du vara lite mer specifik?
risker	produkt- eller projektrisker som inte täckts av övriga punkter	Är alla verkligen så relevanta för testningen? Vad är chefen rädd för?
testutförare	vem/vilka som ska utföra olika delar av strategin	Kan hjälp behövas av andra? Hur samarbetar ni?
testnivåer/testfaser	samlingsbegrepp för testaktiviteter med olika syften	Vore det bättre att göra detta tidigare eller senare?
motiveringar	varför vi gör som vi gör	Har du tillräckligt bra argument?
testidéer	viktiga exempel på vad som kommer testas	Behöver dessa vara med i strategin? Är det något som absolut inte får glömmas?
hur testarna tänker	riktlinjer som guidar testinsatsen	Gör strategin testarna bättre? Vet ni om mångfalden i individers tankegångar?
utelämnat	det som inte kommer testas	Är alla överens om vad som ska utelämnas? Är ni ovetande om saker som utelämnas?
svårigheter	de största utmaningarna för test-projektet	Är du helt ärlig? Börjar ni med att testa det svåraste?
prioriteringar	vad vi kommer lägga mest och minst tid på	Är dessa viktiga beslut genomtänkta, av flera personer?
logistik	testplans-saker som tidsplan, kompetens, testmiljö m.m.	Har du fokuserat för mycket på detta så att strategin blivit lidande?
förklaringar	begrepp och annat som läsarna behöver förstå	Är det tydligt utan att skriva läsaren på näsan? Gör strategin bra reklam för testningen?
rapportering	vilken information som ska förmedlas, och hur	Vem ska använda informationen, och till vad? Behöver någon form av överlämning göras?
		Har du rätt i dina antaganden?
		Blir det för dyrt?
		Är strategin tillräckligt mångfacetterad?
		Kan vi ändra oss när vi lärt oss mer?

Du organiserar strategin på eget sätt, men kan använda dessa kategorier för att se om något viktigt saknas. Ta bara med de saker om är relevanta för ditt unika sammanhang, och lägg till annat som

behövs. Teststrategin ska sammanfatta det viktigaste, och för att få den lättläst och värdefull, så är det viktigt att ha rätt innehåll, att använda **rätt språk**, att lägga sig på lagom hög nivå; men ändå med en detaljrikedom som gör att den är granskningsbar.

Det finns inga regler för hur lång den ska vara, det bör anpassas efter läsarna. Om du misstänker att intressenterna inte kommer läsa ordentligt, applicera Einsteins regel om enkelhet:

> *förenkla så mycket som möjligt men inte mer än så*

Det behöver inte vara fel med två strategier; en på högnivå, och en med viktiga detaljer. Den skrivna teststrategin kan aldrig fånga allt viktigt, så förmedla den muntligt om du har möjlighet, så att frågetecken kan rätas ut med en gång. Och kom ihåg att den första teststrategin aldrig är perfekt; att förbättra efterhand är nästan alltid tillåtet.

Att förklara testningen

Det kan vara på sin plats att i teststrategin berätta lite om vad testningsbegreppen betyder i just er situation, att informativt göra reklam för testningen som kommer att göras. Se upp bara så att det inte blir för stor tyngdpunkt på detta, det är inte ovanligt att IEEE-baserade testplaner innehåller massvis med förklaringar, men väldigt lite om vad som verkligen ska testas, och hur.

> Exempel: Enhetstester skrivs och underhålls av utvecklare och är en central del av strategin för regressionstestning. De skapas tillsammans med koden och säkerställer att komponenten gör det som den är tänkt att göra, Testare deltar gärna som bollplank för ökad täckning. Hos oss lägger vi mest krut för beräkningsmodulerna och databasinteraktionen, där vi vill ha 90% kodtäckning, och acceptans från en granskande kollega.

Genom att berätta om hur testningen går till, och vad den syftar till, så kan detta vara ett sätt att göra viktig PR för testgruppen.

Barnum-strategier

Många teststrategier är, likt Barnum-påståenden, så generella och vaga att de skulle passa på vilken produkt som helst.

> Exempel: Vi kommer testa all ny funktionalitet så pass djupt som vi hinner, och den gamla funktionaliteten mer översiktligt.
> Vi kommer framför allt använda specifikationer och de levande riskanalyser som tas fram.
> Om det blir tid över, så kommer vi göra automatiserade tester för regressionstestning.
> Resultaten kommer rapporteras löpande till de som berörs.

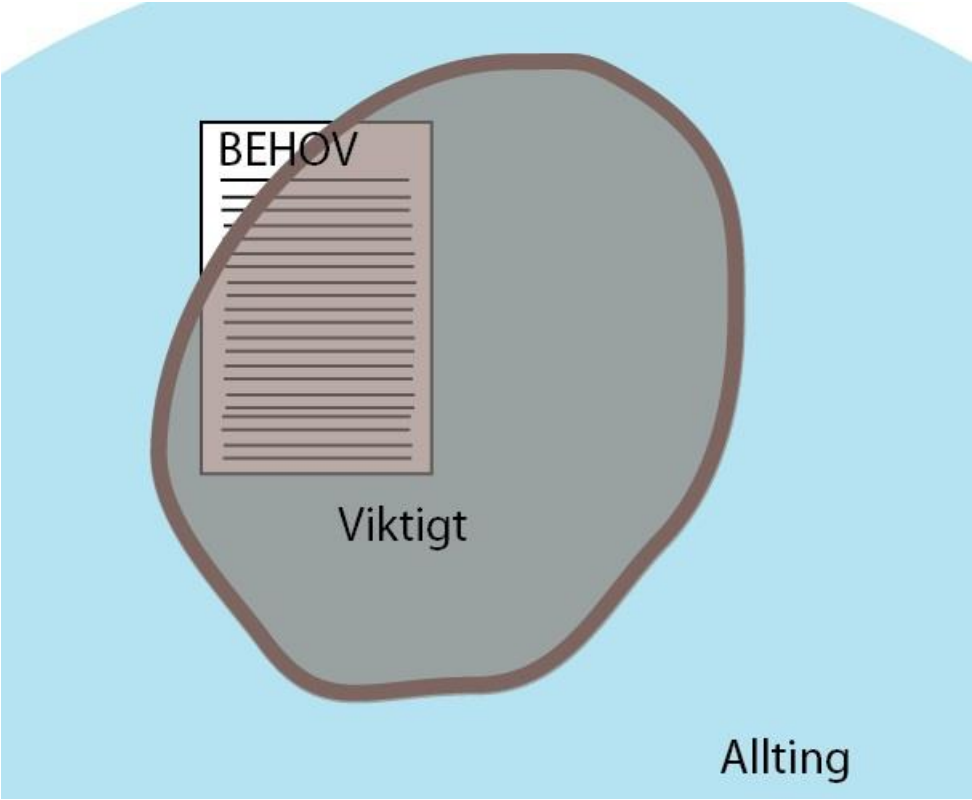
Det är lätt att säga “det ser vettigt ut” om denna, men egentligen är den helt innehållslös och därmed inte värdefull. Ibland har man tagit strategin från förra versionen, som togs från gången innan o.s.v. Ibland har exempeltexten från RUP fått vara kvar, och ibland låter man Approach-avsnittet i IEEE 829 vara det minsta (trots att det troligen är det viktigaste.)

> Kontrollpunkt: Om man kan byta ut produktnamnet mot ett annat, och det fortfarande låter rimligt, så är din strategi troligen inte tillräckligt specifikt, inte tillräckligt “på riktigt”.

Mångfacetterad teststrategi

Den troligen allra viktigaste tumregeln finns beskriven i lektion 283 i *Lessons Learned in Software Testing*: Diverse Half-measures: Det är bättre att testa hyggligt på många sätt, än perfekt på ett eller två.
Orsaken till detta är att testning är en stickprovsaktivitet, vi kan inte testa allt, och om vi försöker göra det, så kommer vi bara täcka en liten del av helheten.

Nedanstående bild kallas programvarupotatisen:



Rektangeln symboliserar kraven, det vi trodde var viktigast innan projektet startade, och som gick att skriva ner på ett bra sätt. Den blå ytan är allt möjligt användande, med allehanda data, omgivning, känslor och behov. Det är en oändlig yta, men vi har tur! Den bruna ytan symboliserar det som är det viktiga.

Det är kring det viktiga vi vill göra det mesta av vår testning, och vi behöver använda många olika testmetoder för att inte snöa in på bara en liten, liten del (det är mycket som är viktigt med programvara!)

När man gör sina stickprov, så uppmuntras att se sig runtomkring för att upptäcka mer. Då inträffar ofta serendipitet - att leta efter något, men hitta något annat som är viktigt. Det kryllar av serendipitet i testning, och det är inget att skämmas för. Snarare ska man skapa många möjligheter till detta, genom att vara beredd på det, genom att förbereda testerna på bra sätt (ny plattform, komplex data), genom att känna till mycket om systemet, och ta chanserna att lägga till en liten avvikelse när det känns befogat.

Din teststrategi består av flera olika delstrategier, och ibland fångar en strategi flera olika saker som är viktiga. Ibland behövs flera olika sätt för att fånga en aspekt, som i nedanstående exempel kring prestanda.

Strategi-exempel: Test-uppdraget är inriktat på prestanda, som behöver utvärderas på flera sätt. Med LoadUI kommer vi testa oss fram till den brytpunkt när lasten blir för stor, med samma verktyg ska vi rapportera svarstider för den normala belastning som finns beskriven i user story 1 och 2. Dessa tekniska, “objektiva” siffror ska kompletteras med upplevd prestanda (testas manuellt) vid ingen, normal, och hög last. Med hjälp av de tre stora webb-läsarnas prestanda-plug-iner hoppas vi hitta front-end-problematik som undgått utvecklarna hittills. Vi behöver tillgång till monitorerings-verktyg för servern, och skulle vilja att en utvecklare sitter med några dagar för att se hur produkten beter sig, och leta efter finjusteringar av inställningar. Vi skulle även vilja ha tillgång till en extra server, som kan stå och tugga under belastning i en vecka (utvärderar uthållighet.) Vi kommer kasta ett getöga på funktionalitet som fungerar sämre i vissa fall, samt användbarhetsproblem som är för uppenbara att ignorera.

Testplaner

Teststrategin kan ses som en delmängd av testplanen, som även innehåller viktiga saker kring hur strategin ska realiseras i verkligheten. Ibland läggs ohygglig tid på testplaner, när man i många sammanhang troligen kunnat nöja sig med lättviktig dokumentation av teststrategin. Vad man ska göra och när, det sköts på samma sätt som resten av projektet. Planering är viktigt, men dokument som ingen läser har man ingen större nytta av.

Gemensamma strategier

Den här boken är mest inriktad på specifika projekt och uppdrag, men i andra sammanhang kan teststrategi betyda de övergripande tankarna som styr många projekt i många releaser. Där kan man placera centrala tankegångar som man inte ständigt vill upprepa

Definitioner; acceptanstestning betyder aldrig samma sak, så att definiera företagets egen användning av termer kan undvika missförstånd.

Kvalitetsmål; förståelsen för vad som är viktigt sägs sitta “i väggarna”. Men tar man sig tid att skriva ner vad kvalitet vanligtvis betyder för oss, så har man bra utgångspunkter både för utvecklare och testare. En bra start får man om man gör relevanta kvalitetsegenskaper specifika för systemet.

Produktspecifika snabbtester; Det tog oss nio år på ett företag att inse att alla inte hade samma små knep för att testa bredare och bättre, och att en gemensam lista skulle vara ett snabbt sätt för nya att komma upp i bra fart. Det tog mindre än en dag att producera en användbar lista.

Riktlinjer; ofta har man tankesätt som hjälper testarna att se det som är viktigt. Det finns säkert situationer där det kan skada att kommunicera dessa outtalade riktlinjer, men det lär inte vara ofta.

Feedback

Be om ärlig feedback, annars kommer du nog inte få den. Att diskutera delar av strategin innan den är klar borgar för mer välgrundade idéer. Oavsett om du kommer få feedback från insatta personer, så kan det vara nyttigt att granska dokumentet för sig själv. Förutom att begrunda alla kapitel i den här boken, så kan du ställa dig den trevliga frågan:

Vad kommer hyllningskörerna gilla allra mest?

Förutom ett förbättrat självförtroende kan detta ge förståelse av vad som är viktigt och kan stärkas ännu mer. Mindre kul, men kanske desto nyttigare är:

Vad kommer den värsta kritikern säga?

Fundera på svaren utan att gå i försvarsställning, och du har god chans att se verkliga brister i din strategi.

Kvalitetssäkring

Förutom att låta intressenter ge återkoppling på strategin, så kan du dubbelkolla att den fångat det viktiga på olika sätt:

* gå igenom relevanta kvalitetsegenskaper, fångas de i din strategi?
* gå igenom intressenters behov, kommer de mötas?
* fundera på vad testare ofta gör hos er, nämns det i strategin?
* fundera på vad ni länge velat prova

Det som är viktigt, är troligen värt att explicit ta med i teststrategin.

Heureka – Teststrategi

Version 1.1

Henrik Emilsson 2009-04-09

1 Testuppdrag

För att vi ska kunna säkerställa att vi testar på rätt sätt och att vi fokuserar på rätt områden, bör vi utforma ett testuppdrag så att alla tänkta testaktiviteter bidrar till att nå projektets mål. Och eftersom testning är en *"teknisk undersökning utförd för att ta fram kvalitetsrelaterad information"* så är det bäst om testningen kan anpassas efter de *kvalitetsmål* som finns.
Vid en undersökning av hur Heurekas projektmedlemmar såg på olika kvalitetskriterier så blev resultatet att de tycker att de högst prioriterade kvalitetsmålen som gäller för Heurekas applikationer ska vara, i prioriterad ordning:

1. Tillförlitlighet – fungerar bra och motstår fel, krasch- och buggfritt, data är säkerhetskopierad.
2. Precision – räknar rätt.
3. Användbarhet (Användarvänlighet) –intuitivt och lätt att använda.
4. Modifierbarhet – bra möjligheter att bygga ut med fler funktioner och göra ändringar.

Dessa mål, i samma prioritetsordning, verkar vara överensstämmande både när medlemmarna blev ombedda att rangordna de tre viktigaste kvalitetskriterierna (utifrån olika intressenters behov) samt när de blev ombedda att rangordna varje kvalitetskriterium individuellt. (För mer information om undersökningen, se resultat och vidare hänvisning i kapitel 5.1 - Kvalitetsundersökning)
Givet de kvalitetsmål som gäller för projektet så är testuppdraget följande, i prioriterad ordning:

- Hitta allvarliga fel som måste fixas innan systemet går i produktion.
- Säkerställ att beräknad numerisk data är korrekt.
- Identifiera problem i design som påverkar systemets användbarhet.
- Skapa en svit av regressionstester för att underlätta framtida modifieringar av systemet.

2 Teststrategi

Teststrategin beskriver sambandet mellan testprojektet och testuppdraget och handlar om vad vi ska testa och på vilket sätt vi ska göra det.
Denna strategi är baserad på rådande situation och sammanhang som gäller för projektet och är avsedd att användas som en vägledning för den testinsats som projektet kräver.
Om situationen kräver det, eller om yttre förändringar har större påverkan på projektet, kan det finnas behov av att förändra och förbättra denna strategi under projektets gång.

2.1 Teststrategi – Sammanfattning

2.1.1 Ökad tillförlitlighet

För att öka tillförlitligheten till Heurekas applikationer kommer vi att behöva göra ett antal insatser som sammantaget ger oss en chans att hinna förbättra tillförlitligheten:

- Hitta allvarliga fel som måste fixas innan release;
- Hitta fel som förhindrar användaren att utföra sina uppgifter;
- Undersöka områden där Heureka potentiellt sett kan skada eller tappa bort data;
- Identifiera områden i produkterna där den samlade bilden av rådande design och/eller ett antal små problem ger att förtroendet för applikationen kan skadas.

Vi är även intresserade av att försöka hitta så många fel som möjligt.
Syftet med detta är delvis för att kunna fixa dem innan systemet går i produktion, men detta görs också för att själva kunskapen om de allvarliga felen medgör att projektledningen kan fatta medvetna beslut om vilka åtgärder som bör tas.

2.1.2 Korrekta beräkningar

För att säkerställa att beräkningar är korrekta kommer vi att först behöva skriva enhetstester för att

verifiera varje enskild beräkning. Dessa enhetstester kommer vi sedan att granska utförligt för att säkerställa att både tester och kod gör rätt sak.
Och för att kontrollera att beräkningarna är rätt när de har kopplats ihop, kommer vi att behöva skapa ett mindre antal tester på systemnivå där vi verifierar att integrationen av beräkningarna är korrekt.
I och med att vi fokuserar på att varje enhet gör rätt så behöver vi inte göra en fullödig test på systemnivå; det blir både enklare och billigare att göra dem på enhetsnivå.
Enhetstesterna görs av den utvecklare som är ansvarig för koden; dock kan en systemtestare med fördel bidra till vilka tester som bör tas fram.
Enhetstesterna ska verifiera den modell som tagits fram för projektet, samt använda den medföljande testdatan som verifieringsdata.

2.1.3 Identifiera designproblem

För att förbättra användarupplevelsen och produkternas användbarhet så kan testningen bidra med att ta fram kvalitetsrelaterad information om de *användbarhetsproblem* som hittas i systemet.
Det kan gälla såväl de fel som uppkommer vid daglig användning och som därmed förhindrar användaren att utföra sina uppgifter, men det kan också innebära att vi belyser de områden där vi uppfattar att användare har problem med att lösa uppgifter.

2.1.4 Regressionstest

För att kunna hantera nuvarande system samt framtida förändringar och modifieringar i systemet så kommer vi att ta fram en svit av *scenariotester* som i sin tur bildar en regressionstestsvit.
De scenarier som ska ingå i sviten ska vara trovärdiga och realistiska och som är sådana fall där det alltid måste fungera.

2.1.5 Testperioder

Förutom de fokusområden som har nämnts ovan, så kommer vi även att testa iterativt under projektets gång. Dvs att efter varje officiellt bygge så körs en testomgång av systemtester.
Systemtestning kommer att utföras kontinuerligt och alla fel som hittas kommer att rapporteras så snart som möjligt och på överenskommet sätt; alla problem som uppstår kommer att flaggas upp så fort som möjligt.

2.2 Teststrategi – Detaljer

Nedan följer detaljerade beskrivningar av de åtgärder som inte bara ligger i linje med teststrategin utan ska också se till att vi fullföljer testuppdraget.

2.2.1 Enhetstester

Här listas de åtgärder som bör göras med avseende på enhetstest.

2.2.1.1 Dela upp enhetstester i två klasser

Enhetstesterna bör delas upp i två typer som baseras på komplexitet och körtid:
enhetstester och **beräkningstester** (eller integrationstester).
Detta har flera fördelar, men framför allt så tar det mindre tid att köra igenom enhetstester innan och efter man har checkat in kod. Detta gör att man som programmerare blir mer benägen att köra unittesterna som det hjälpmedel de faktiskt utgör.
Och å andra sidan kan man då köra de tyngre beräkningstesterna vid särskilt utvalda tillfällen, t.ex under helger, vid större incheckningar, vid bygge, etc. Dvs vid de tillfällen då man har en ökad tolerans och acceptans för att testerna kan ta längre tid.
Det finns, som jag ser det, två bra exempel på hur man kan dela upp testerna i de två typerna.
Den ena exemplet är att separera ut beräkningstesterna till egna projekt eller namespace i Visual Studio. På så vis kan man fortsatt använda Testrunner.net för att köra tester inifrån Visual Studio på samma sätt som idag. Dock innebär detta att man måste bryta ut tester och skapa nya filer.

Det andra exemplet är att använda sig av NUnits kategorier och den nya funktionen i Testdriven.net (v.2.17) som innebär att man kan köra testerna baserat på kategori. Se mer här: http://weblogs.asp.net/nunitaddin/archive/2008/12/03/testdriven-net-options-pane.aspx

2.2.1.2 Beräkningstester

Beräkningstesterna ska verifiera att koden motsvarar den modell som tagits fram för projektet, samt använda den medföljande testdatan som verifieringsdata.
Detta innebär att vi då hänvisar till den modell som finns inlagd i Modelldatabasen, samt att vi kan lägga till den testdata vi har utgått ifrån och också behandla den som en officiell part av modellbeskrivningen.
På detta sätt får vi också ett tydligt sätt att kommunicera de förväntningar som finns på att beräkningarna är rätt; vi kan då peka på att implementation har gjorts enligt modell X - version Y.Y, och testerna är baserade på testdata X.Z.
Om det saknas testdata, eller beskrivning i modellen, för andra trädtyper än de ursprungliga så medför detta att vi även här har något att utgå ifrån då vi behöver bestämma hur vi ska hantera de resterande trädtyperna och beräkningarna för dem.
För att säkerställa att beräkningar är korrekta kommer vi att försöka följa denna procedur:

- Först skriva enhetstester för att verifiera varje enskild enhet kod;
- Sedan skapa beräkningstester för att verifera enskilda beräkningar;
- Dessa beräkningstester kommer vi sedan att granska utförligt för att säkerställa att både tester och kod gör rätt sak enligt den modell som ska finnas som förlaga;
- Och för att ytterligare kontrollera att beräkningarna är rätt när de har kopplats ihop, kommer vi att behöva skapa ett mindre antal tester på systemnivå där vi verifierar att integrationen av beräkningarna är korrekt.

2.2.1.3 Vem gör vad

Enhetstesterna görs av den utvecklare som är ansvarig för koden.
Beräkningstesterna görs primärt av den utvecklare som är ansvarig för koden; dock kan vi med fördel låta någon annan göra testerna för att få en större objektivitet.

2.2.2 Systemtest

Här listas de åtgärder som bör göras med avseende på systemtest.

2.2.2.1 Systemtest vs. Acceptanstest

Systemtest är ett ganska brett begrepp och innebär egentligen bara att testningen sker på ett fullt system, eller på systemnivå. Vad vi menar med systemtest är egentligen *Black Box Systemtest*, men jag tycker att vi förkortar det till *Systemtest* för enkelhetens skull.
När vi utför systemtest så har vi målet att undersöka om den implementerade funktionen/modulen/etc fungerar själv och ihop med andra delar av applikationen. Men detta innebär också att vi ska ha inställningen att **vilja hitta fel**.
Skillnaden blir tydlig om man jämför med att utföra *acceptanstest*, där alltså målet snarare är att verifiera att det fungerar enligt de kriterier man har satt upp. Även om Acceptanstest utförs på systemnivå så räknar vi alltså inte in det när vi menar Systemtest.

2.2.2.2 Test av egen implementerad kod

Checklistor bör tas fram för utvecklarna så att man själv kan testa igenom sin implementering och därigenom själv hitta och fixa buggar tidigt. Dessa checklistor ska inte lista färdiga tester utan ska snarare användas som ett hjälpmedel för att hitta bra tester att köra, men kan också trigga idéer för vidare testning.
Ett förslag på en checklista skulle kunna vara Elisabeth Hendrickson's "Test Heuristics Cheat Sheet",
http://testobsessed.com/wordpress/wp-content/uploads/2007/02/testheuristicscheatsheetv1.pdf

2.2.2.3 *Test av ny funktionalitet*

Vid test av ny funktionalitet så kan man med fördel använda sig av checklistan som kommer att finnas. Men jag vill rekommendera att man försöker köra den nya funktionaliteten tillsammans med ”gammal”. Detta för att hitta fel som kan upptäckas vid systemtest.

2.2.2.4 *Hitta allvarliga fel*

För att hitta allvarliga fel som måste fixas innan release så bör man testa applikationen med intentionen att vilja hitta fel, allvarliga fel.
Då passar det bäst att utföra testningen enligt en metod som kallas *risk-baserad testning* (http://www.testingeducation.org/BBST/BBSTRisk-BasedTesting.html).
Detta går i korthet ut på att identifera, eller försöka tänka ut, hur en applikation kan fallera; för att sedan designa tester för att se om applikationen kan fallera på detta sätt.

2.2.2.5 *Hitta användbarhetsproblem*

För att hitta de fel som uppkommer vid daglig användning och som därmed förhindrar användaren att utföra sina uppgifter, så ska vi fokusera på att försöka hitta de fel som uppträder (eller kan uppträda) under ett scenario som är vanligt hos våra applikationers användare. Förhoppningsvis kan vi identifera de flesta sådana fel i de scenario-tester som vi tar fram; och därmed hinna fixa de vi vill innan release.
Detta innebär inte att vi ska fixa alla fel som vi hittar; det innebär inte heller att vi ska leta triviala fel ute i periferin. Snarare handlar detta om att hitta de fel som de flesta av våra användare har en chans att hitta och sedan gör vi bedömningen om vi behöver fixa dem eller inte.
Ytterligare ett sätt att förhöja användbarhet och användarens upplevelse är att skapa riktlinjer för hur vi tar fram och designar grafiska gränssnitt.
Ett exempel är att följa *”Windows User Experience Interaction Guidelines”* (http://msdn.microsoft.com/en-us/library/aa511258.aspx).

2.2.2.6 *Undersök dataintegritetsområden*

Att undersöka områden där Heureka potentiellt sett kan skada eller tappa bort data innebär att utföra *risk-baserad testning*. Egentligen är ju denna punkt en underpunkt till kapitel 2.2.2.4 - Hitta allvarliga fel, men jag tycker att fokus här är mer fokuserat på att hitta problem där data kan skadas eller tappas bort.
Här bör vi fokusera på att försöka identifera områden i Heurekas applikationer där datan på något sätt skulle kunna ta skada eller försvinna. Exempelvis, vad händer om databasen går ner under tiden man arbetar i ett projekt, vad händer om databasen går ner när man skriver till den, om man förlorar nätverkskontakt under interaktion med databasen, etc

2.2.2.7 *Identifiera riskområden*

Identifiera områden i produkterna där den samlade bilden av rådande design och/eller ett antal små problem ger att förtroendet för applikationen kan skadas.

2.2.3 Installationstester

För att skapa en strukturerad testning av ett nytt bygge så bör vi skapa en installationstest-svit där vi på ett organiserat sätt kan genomföra tester av varje bygge enligt ett visst överenskommet sätt.
Detta medför att vi bör fånga eventuella problem som kan dyka upp från en version till nästa; och det ger oss också en säkerhet om hur programmen beter sig vid installation.
Testerna bör ske på en dator som inte har någon utvecklingsmiljö installerad; och det bör också vara en dator där vi kan skapa ett antal olika användare med olika behörighet.

2.2.4 Regressionstester

För att kunna hantera nuvarande system samt framtida förändringar och modifieringar i systemet så kommer vi att ta fram en svit av scenariotester som i sin tur bildar en regressionstestsvit.
De scenarier som ska ingå i sviten ska vara trovärdiga och realistiska och som är sådana fall där det

alltid måste fungera. Dessutom kommer vi att bygga ut med ytterligare några alternativa scenarier där vi sammantaget försöker fånga så många varianter som möjligt.
Troligtvis kan vi använda oss av verktyget PICT (Pairwise Independent Combinatorial Testing) för att kombinera ihop ett antal scenarier som täcker in så många kombinationer som möjligt.

2.3 Riktlinjer/Policy

Vi bör tillsammans komma överens om definitionerna och beskriva följande viktiga begrepp och händelser i projektet:

- ”Redo för test”
- ”Testat”
- Byggprocedur
- Unittest-strategi
- Beräkningstest-strategi
- Systemtest-strategi

2.3.1 Definitioner

Interna definitioner av begrepp.

Klart för test	Redo för systemtest i utvecklingsmiljön. • Skrivit kod • Koden är dokumenterad. *Syfte: Att en programmerare ska kunna underhålla koden.* • Skrivit enhetstester *Syfte: Verifiera att den kod som har skrivits gör rätt sak; samt att underlätta vid framtida förändringar av kod.* • Enhetstester är ”godkända” av någon utvecklare i teamet. *Syfte: Att se om testerna testar rätt saker.* • Databas-script är klara.
Redo för release	Inkluderas i nästa release. • Testat eller kontrollerat av Henrik, Peder, Beställare eller Utvecklare. *Syfte: Hitta fel och problem så tidigt som möjligt.*
Testat	Inkluderat i aktuell release. • Godkänt av beställare. *Syfte: Få ett kvitto på att implementerad funktionalitet är tillräckligt bra.* • Acceptanstest utfört. *Syfte: Verifiera att vi har gjort rätt sak.*
Release	Ett bygge av en produkt med: • tillhörande versionsnummer • release notes • databas-script • publicering

2.3.2 Exempel

todo

3 Logistik

Detta kapitel är avsett för att beskriva de resurser som behövs för att förverkliga teststrategin.

todo.

4 Ordförklaringar

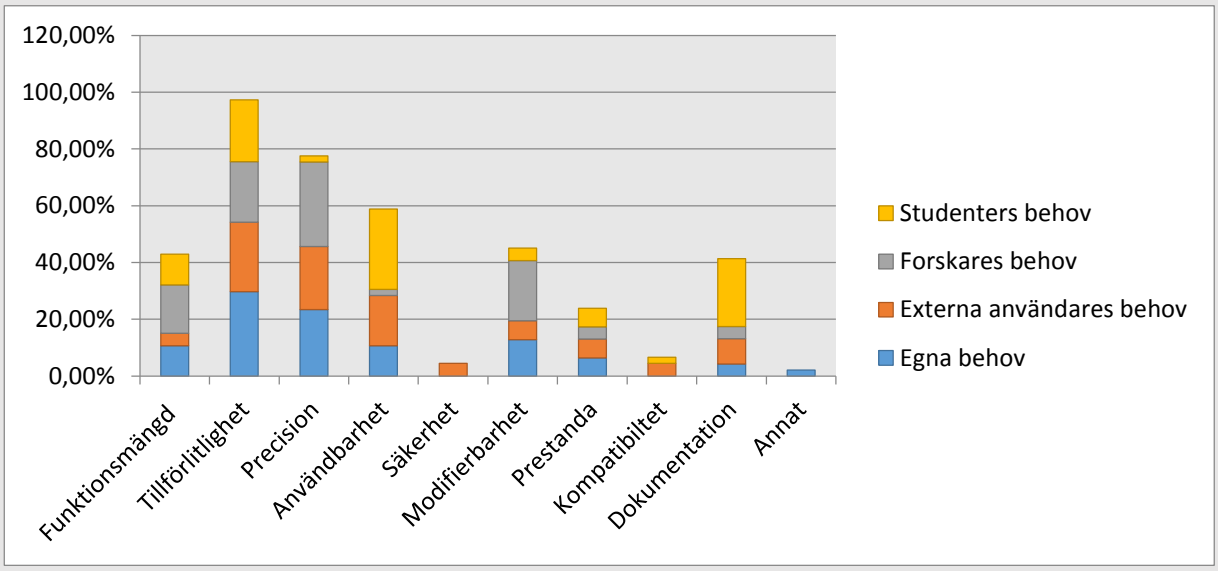
Förklaringar och defintioner av ord och begrepp som används i detta dokument; samt som ett förtydligande där flera betydelser kan finnas om vad vi avser i vår kontext.
Tanken är att vi fyller på denna lista efter hand som frågor dyker upp.

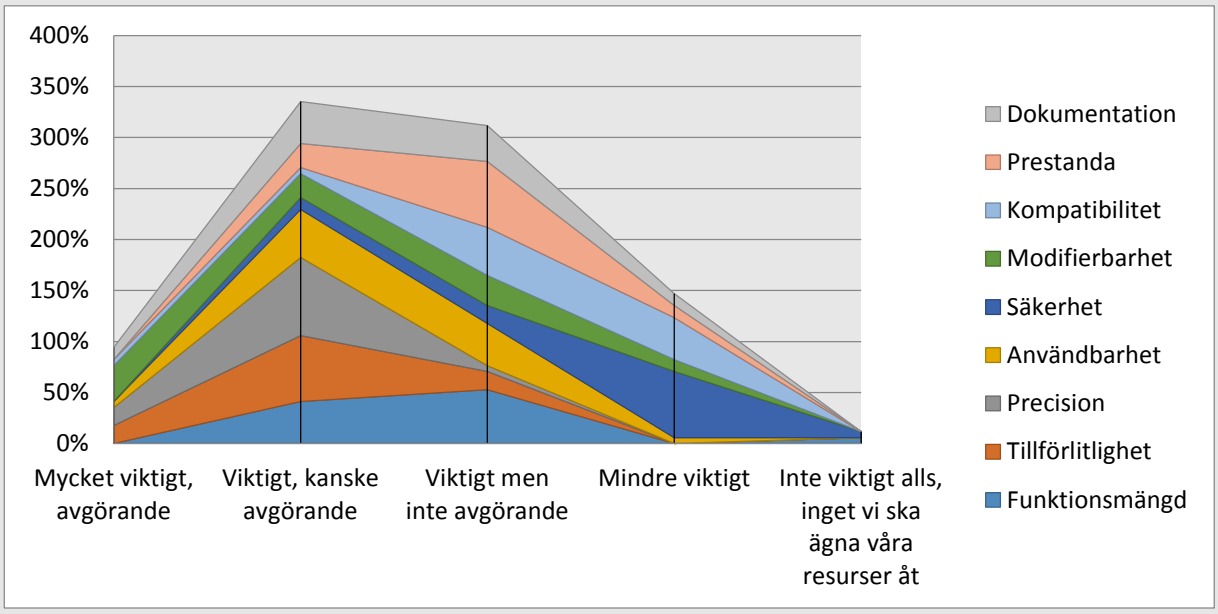
Ordförklaringar 4-1

Systemtest	Testning utförd på systemnivå, dvs testning på ett körbart system där alla komponenter har integrerats och byggts ihop. Systemtestning är (oftast) Black box. http://en.wikipedia.org/wiki/System_testing
Enhetstest	Testning utförd på isolerade delar av koden, dvs verifiering av individuella enheter av kod. En enhet är den minsta testbara delen av en applikation. Enhetstestning är White box. http://en.wikipedia.org/wiki/Unit_testing
Black box testning	Testning utan vetskap om intern struktur i en applikation. http://en.wikipedia.org/wiki/Black_box_testing
White box testning	Testning med vetskap om intern struktur och bakomliggande kod. http://en.wikipedia.org/wiki/White_box_testing
PICT	Pairwise Independent Combinatorial Testing http://download.microsoft.com/download/f/5/5/f55484df-8494-48fa-8dbd-8c6f76cc014b/pict33.msi
Risk-baserad testning	En teknik inom fältet Black box systemtestning. Det går ut på att försöka tänka ut hur en applikation kan fallera; för att sedan designa tester som triggar dessa fel. http://www.testingeducation.org/BBST/BBSTRisk-BasedTesting.html
Scenariotestning	Scenariotester är verklighetstrogna, trovärdiga och motiverande tester för intressenter; utmanande och komplexa för programmet; och lätta att utvärdera för testaren. http://www.testingeducation.org/BBST/ScenarioTesting.html
Regressionstestning	Testning utförd för att kontrollera att tidigare fungerande funktionalitet inte har gått sönder. TODO! http://www.testingeducation.org/BBST/RegressionTesting.html

5 Appendix

5.1 Kvalitetsundersökning





Högnivåstrategi PDF Export i Word

Här följer en högnivåstrategi som tagits fram för ett påhittat scenario att hjälpa Microsoft vid planering av testning av den nya versionen av PDF Export för Word. Notera att det enda vi vet är att Microsoft har blivit rättsligt tvingade att köpa in funktionen Exportera till PDF av en extern leverantör, (för att de inte har följt Adobes riktlinjer.)

Vi inte har tillgång till all information, så detta är bara en start som behöver vidareutvecklas, där en del av idéerna möjligen är alltför ambitiösa.

Automatisering

Då miljontals användare med unik data kommer använda denna funktion, så behövs en stabil grund för automatiserade tester. Om denna redan finns, så bör det säkerställas att den klarar följande:
* automatgenerering av ohyggligt stort batteri med testfiler (testar funktionen, och stabilitet)
* kan köras på alla olika språk för att fånga lokaliseringsproblem
* kunna lagra PDF-resultat för framtida regressionstestning
* kan validera att genererade PDF:er inte bryter mot PDF-standard
* kunna återanvändas för PowerPoint och andra program som också ska använda samma komponent
* automatgenerera slumpad data i Word, så att vi över tid får väldigt god täckning på olika bokstäver, kombinationer, bilder, tabeller m.m.
* mäta tidsåtgång så att prestandaproblem snabbt kan identifieras
* kunna köras på olika plattformar

Krav-baserad testning

Kraven kan delas upp i två delar; de som externa leverantören implementerat och testat, och de systemkrav som Word 2015 PDF Export ska uppfylla.
Krav som är lätta att automatisera, eller måste köras väldigt ofta, bör automatiseras.
Övriga krav kan testas manuellt utifrån specifikationerna, gärna i par för att få fler ögon.

Risk-baserad testning

Bugghistoriken ger en bra bild av vad som lätt kan gå fel för denna funktionalitet, så relevanta problem bör försöka återskapas. Ett axplock av övriga risker:
* resultatet ska vara så likt originalet det kan vara
* rätt filstorlek och bildkvalitet
* korrupta filer
* säkerhet, ta in era bästa pen-testare
* upplevd prestanda är lika viktig som den objektivt mätbara
* att pdf:erna kan användas på många sätt, utskrifter, mobila enheter, andra program etc.
* med mera, utforskande testning av Word-sakkunniga är lämpligt

Användbarhetstestning

Om gränssnittet ska ändras, så bör det göras användartester med olika sorters användare.
En längre Betaperiod skulle kunna användas för den sista finslipningen (bra för kompatibilitet också)
De manuella testarna bör utbildas i Microsofts UI Guidelines, och i de kvalitetsaspekter av

användbarhet som de bör vara på sin vakt mot under testningen.

Standarder

En expert på PDF-standard bör vara involverad i testningen så att vi följer rätt version, men samtidigt är kompatibla med äldre PDF-läsare.
De Adobe-riktlinjer som tidigare brutits mot måste analyseras noggrant, så att det inte blir samma rättsliga besvär en gång till.

Plattformar

Sätt upp en matris med de viktigaste konfigurationerna, och variera bland dessa under testningens gång.
Förutom datorer och mobila enheter, så ska olika PDF-läsare användas.
Troligen har ni ett gäng "best representatives" som innehåller de mest vanliga och felbenägna plattformarna. Dessa bör kryddas med PDF-specifika detaljer.

Lokalisering

Grundfunktionaliteten bör täckas av de automatiska testerna, men kryddas med utforskande testning som kan fånga upp specifika saker för respektive språk.
Språkexperter bör granska texterna även genom att använda programmet (säkerställer att det är bra i sitt sammanhang.)

Integrationstestning

Utan att känna till de tekniska detaljerna, så misstänker jag att den externa leverantören även kommer implementera API:er som kan användas för integrationstestning.
Om möjligt kan det göras tillsammans med automatiserings-insatsen.

Kvalitetssäkring av leverantör

Leverantörens egna tester och resultat bör analyseras.
Leverantörens metoder för kvalitetsarbete (t.ex. kodgransking, säkerhet, PDF-standard) bör undersökas.

Konkurrentanalys

För att behålla den ledande positionen inom textbehandling bör genererade PDF:er vara minst lika bra som konkurrenternas. Här lär ni ha mer material än vad vi kan erbjuda i dagsläget ;)

Avslutning

Detta är en start som vi gärna pratar vidare om med er som kan detaljerna kring Microsoft Office, och PDF.
Hoppas att det varit till någon nytta!

Levande teststrategi

Vi vet att vi inte vet allt

Strategin ska guida vad som testas och hur. Den egentliga strategin som används, är dock respektive persons egen förståelse av teststrategin. Hur tydlig du än är, så kommer alla i projektet ha en unik bild av vilken teststrategi som gäller. Det kan vara en fördel att ha olika mentala modeller, men om man pratar om dem, så kan man undvika missförstånd och problem.

Vi tycker också att den enskilde testaren ska ha tillåtelse att reparera och utöka teststrategin. Om man märker att nuvarande metoder inte ger vettiga resultat, så borde varje testare kunna komma med förändringsförslag.

Förändringar

Den vanligaste förändringen i teststrategier gäller vad som läggs mest tid på. Saker var svårare än förväntat, eller mängden "viktigt" hade underskattats. Detta är ofta viktigt att kommunicera, speciellt i situationer där tidsåtgång har stort fokus redan i testplaneringen.

Förändring är ingenting att vara rädd för, om nuvarande testmetoder inte biter bra, så är det klart att vi ska prova nya sätt.

Att förstå vad som är bra

Det är inte så svårt att komma på många sätt att testa, och den viktiga färdigheten är förmågan att bedöma vad som är bra, eller åtminstone fruktbart. Det är som när man skriver, eller skapar musik, ibland förstår man bara att man träffar rätt (men det är klart att man kan ha fel…) Denna känsla baserar sig på många erfarenheter, på mycket reflektion, samt förmågan att kritiskt granska även sina egna beslut (om man känner att samma typ av strategi alltid är bra, så är det dags att få ny inspiration!)

Och du kan inte veta att en strategi är bra förrän möjligen efteråt...

Exempel

Testuppdrag: Ledningen betonar automatiserad testning.

Skriven teststrategi: "Under projektets gång kommer vi identifiera tester som är lämpliga för automatisk regressionstestning, och dessa kommer implementeras när respektive område är stabilt."
Outtalat tillägg till strategin: (...i slutet, om vi inte har annat som är viktigare.)

Efter halva projektet frågar huvudprojektledaren om han kan få se resultaten av automat-testerna, och på svaret att inga tillverkats än, så blir det sura miner.

I det här fallet så borde ledningen berättat mer om varför de vill automatisera (det är väl inte bara något de läst någonstans...) och testarna borde berättat om vilka kriterier de har för att bedöma när det är fruktbart att automatisera. Man borde prata om hur man väger automatiseringsuppdraget mot "hitta viktiga problem".

Man har dock inte tid, eller förmåga, att reda ut allt i början. Saker måste få växa fram, och möta de oväntade saker som alltid händer. Dessutom kommer din teststrategi med tiden bli sämre och sämre eftersom omgivningens behov, och teknologierna, förändras.

Dokumentationsnivå

Det är sällan värt att ha en dokumenterad teststrategi som alltid är uppdaterad, men du behöver kommunicera viktiga ändringar på lämpligt sätt.

För små ändringar kan det mycket väl räcka med att det manifesteras i de tester som utförs.
För det är väl resultatet som räknas, inte sant?
Och det är kommunikationen av resultatet som är värdefullt.

Rapportering

Testningen är aldrig bättre än kommunikationen av resultaten

Det är inte ovanligt att testgrupper upplever att ledningen inte förstår deras situation, att det ställs orimliga tidskrav, och att de som bestämmer inte lyssnar.

Hur man rapporterar kan vara en orsak, men problemet kan ligga tidigare: man har inte förankrat sin teststrategi. Rapporter misstolkas och uppskattas inte, för man vet inte vad testarna försöker göra, vad de tror att de måste göra. Intressenterna har aldrig fått chansen att förtydliga testuppdraget.

Så hur rapporteringen går till är kanske inte det viktigaste (men lyssna till hur mottagaren vill ha det), utan det viktigaste är att se till att ledningen håller med om teststrategin. Då kommer man förstå resultaten, speciellt om man har en delad kvalitetsmodell med egna, signifikanta och värdeladdade ord.

Det är fortfarande en utmaning dock; hur berättar jag om täckning på bästa sätt, hur tydliggör jag skillnader mellan produktstatus och projektstatus, stämmer informationen med det som fanns i ditt uppdrag? Hur kan jag rakryggat yppa sanningen och se till att informationen når fram?

Avvikelser är normalt, vi lär oss ju saker, men har fokus varit rätt? Är det läge att ärligt berätta hur bra du tycker att testningen varit?

Kontrollpunkt; fundera på dessa frågor:

Varför rapporterar du?
Vad är viktigast?
Till vem rapporterar du?
Passar språket läsarna?
På vilka olika sätt rapporterar du?
När är det lämpligast att rapportera?
Får du nya testidéer när du rapporterar?
Rapporterar du bara för att du måste?
Rapporterar du det viktiga eller det du tror att mottagaren vill höra?

Snedvridningar

Du kan inte undvika tankefällor, men du kan hantera dem

Våra hjärnor är fantastiska, men kan ibland ta tveksamma beslut på grund av snedvridningar; tankefällor som är lätta att glida in i. Det finns mycket forskning och litteratur om detta, men ännu inget specifikt för testning. Detta kapitel har författats under tiden som Kahnemans *Tänka, snabbt och långsamt* lästes, och varje underkapitel kommer från boken, men nu i teststrategins värld, baserat på misstag vi sett, och själva gjort.

Svara på enklare frågor

“Hur ska vi kunna testa allt som är viktigt?” är en väldigt svår fråga. Det är mänskligt, men tvivelaktigt, att byta ut detta mot frågor som är lättare att besvara:

Hur ska vi testa den nya funktionaliteten?
Hur ska vi göra samma testning som förut, men snabbare?
Vad kan automatiseras?
Hur lång behöver testplanen vara?

När vi sedan besvarar den enklare frågan, så tror vi omedvetet att vi även svarat på den svårare frågan... Detta sker också utan att vi är medvetna om det, till exempel när man ombeds tidsuppskatta en testningsinsats innan man funderat över vilka teststrategier som är vettiga. I nästa steg kommer vi sedan att välja den strategi som ger det resultat som tidsuppskattningen hamnat på. En annan variant är att vi fokuserar på att dela upp problemet i mindre bitar, exempelvis testfaser, där vi fördelar ut ansvaret i mindre portioner, med resultatet att helhetsfrågeställningarna och massa annat trillar mellan stolarna.

Motmedel: Sök upp de svåra frågorna. För varje del av teststrategin, överväg om svaret (eller frågan!) är för snäv. Var speciellt vaksam på att utnyttja hur en specifik strategi kan användas för fler informationsmål än originalet.

Exempel: När manuell testning utförs, så varierar man plattform som används, så att en speciell kompatibilitets-fas inte behövs. Vid lämpliga tillfällen görs riktade insatser mot plattformar som behöver täckas.

Det du ser är allt som finns (WYSIATI)

Det är mänskligt att basera sina beslut bara på det vi ser, för att vi tror att det är allt som finns.

För testning kan detta inträffa om man bara ser till testarnas (och utvecklarnas) testning när helt andra strategier kan vara mer gångbara. Men även åt andra hållet; att kunder gör acceptanstestning behöver inte betyda att ingen annan ska utvärdera om produkten är “bra”.

Det kanske vanligaste misstaget, man "ser" kraven, och tror att det är det enda viktiga att testa.

Motmedel: Diskutera vilka mer saker som behöver beaktas, inte bara det som står i projektplanen.

Exempel: Vi kommer ta in personer från andra avdelningar då och då, för att få nya perspektiv, och fruktbara samtal.

Haloeffekten

Om någonting är bra eller dåligt, så överför vi gärna de egenskaperna på andra, egentligen okända områden. Exempelvis kan en lång och stilig testplan får oss att tro att innehållet är vettigt, och en slarvigt uttryckt buggrapport får oss att tvivla på innehållet.

Teststrategimässigt tror vi att den allra största negativa effekten fås när produktens beteende på ett område för mycket påverkar våra förväntningar på andra. När något specifikt ser bra ut, så kan vi grunda vår strategi på att resten också är bra, och om vi ser en kraschbugg direkt, så kan vi tro att fortsatt testning är lönlös, produkten är ej testbar.

Motmedel: Basera inte viktiga beslut på enstaka observationer; titta lite till, ha inte förutfattade meningar om huruvida produktens kvalitet är bra eller dålig (låt observationerna avgöra det.)
Distribuera strategin i textformat, så att innehållet får fokus.

Tillgänglighetssnedvridning

Tillgänglighetssnedvridningen, "*framför allt värderar vi ofta en tankes betydelse utifrån hur lätt den dyker upp*" kan tillföra en del vad gäller mångfacetterade teststrategier.

De teststrategier som först dyker upp (det vi är vana) tror vi är bäst.

* vi minns ett bra exempel, och kör på samma strategi igen
* vi listar många risker, och tror att systemet är säkrare
* vi är på dåligt humör på grund av stress, och vågar inte ta nya beslut

Motmedel: gå inte på rutin, utan diskutera och fundera på alternativa testmetoder.

Validitetsvillan

Du kör samma strategi som förra gången, för det hittades många viktiga buggar. Att lika många missades, och att det kostade ofantligt mycket, det struntar vi i. Ett exemplar räcker för att rättfärdiga, 1 viktig bugg efter att ha undersökt 100 bakåtkompatibilitetsfiler gör det mödan värt (vilket kan vara sant, men det är inte säkert.) Denna gäller även tvärtom, du tror att det är ogiltigt för att det inte gav resultat förra gången.

Motmedel: Även om vi är experter, så kan vi inte förutspå framtiden, därav ytterligare en anledning till mångfald i teststrategin.

Exempel: Eftersom vi kört fri utforskande testning i de senaste regressionstestfaserna, så kommer vi nu köra varje funktion för sig, utefter kravspecifikationen.

Optimistiska snedvridningar

Våra planer är allt som oftast optimistiska; vi föreställer oss på topp varje dag, i lätt nedförslut med solen i ryggen och medvind. Så blir lätt våra teststrategier också, de siktar högre än vi klarar,

speciellt för att det alltid händer oväntade saker. Därför kanske det är än mer viktigt att i sin teststrategi informera om vad som kommer läggas mest tid på, och vilka delar som kommer få en lättviktig behandling (sällan klokt att strunta helt i relevanta delar av strategin.) Din teststrategi ska vara realistisk, och även ta överdriven optimism, och okända okändheter i beaktande.

> Motmedel: Se till att du informerar om prioriteringar, så att lättviktig testning blir för de mindre viktiga områdena.

Fokuseringsvillan

"Ingen enskild del av teststrategin är så viktig som du inbillar dig när du tänker på den."
Denna faller jag fortfarande in i, speciellt när jag försöker övertyga någon om ett visst sätt att testa. Sanningen är ju den att mycket av testtäckningen för det allra viktigaste överlappar; ett gravt installationsproblem fångas med vilken testmetod som helst, en vaken manuell testare kan se både användbarhets-, prestanda- och säkerhetsproblem, och specialisterna på respektive områden kan också upptäcka på ett brett sätt.

> Motmedel: Avfokusera. Se på helheten, diskutera med personer som inte är besatta av samma saker som du.

Övrigt

En viktig förklaring till tvivelaktiga teststrategier kan vara rädsla för ånger och utkrävande av ansvar. Om man kör samma strategi som förra gången är det mindre risk för klagomål än om man aktivt gjort förändringar. Om man kan hänvisa till en best practice, så kan folk tro att det var förståeliga val som gjordes. Men som med alla andra snedvridningar borde man snarare välja det man tror är bäst, än det som minst riskerar klander.

> Motmedel: Mångfald i teststrategin, och mod att ändra sig när man ser resultaten.

> Exempel: En automatiseringsstrategi har inte gett märkvärdiga resultat på ett år. Ska man plöja in ännu mer resurser eller börja om på ny kula?

Tur nämns ofta av Kahneman, och den är naturligtvis viktig även inom testning. Men bra testare har ofta tur, och en bra teststrategi skapar möjligheter att ha tur. I en stickprovsbransch är serendipitet inget att skämmas för!

Avslutning

Om du har sträckläst, så kan det här upplevas som överdrivet stort. Det behöver det inte vara, din teststrategi kan skapas på fem minuter och förmedlas muntligt. Men vi tror att du kan ha nytta av de olika strukturerna vi visat på, så att det blir lättare att förstå vad som är viktigt, och mindre risk att man missat något.

Det är det det handlar om: förstå vad som är viktigt, ta fram en strategi som kan nå den informationen, ändra dig när du vet mer. Var ödmjuk med testningens roll; det finns många sammanhang som klarar sig bra enbart på den testning som utvecklare och slutanvändare gör (detta behöver inte göra dig arbetslös, du kan hjälpa dem bli effektivare.)

Man ska heller inte glömma att en bra teststrategi bara är en start, det krävs mycket och bra testning för att hitta resultaten, som måste kommuniceras på lämpligt sätt.

Hursomhelst, vi hoppas att du som nått så här långt har fått inspiration till ditt testarbete, och att det har varit läsvärt. Ta nu en titt på referenserna, kanske ger de ytterligare information, och lägg sedan ifrån dig denna lilla bok. Plocka upp den igen om ett år, och om vi har tur kommer du säga: “*Det där kan jag redan.*”

/Rikard & Henrik

Referenser

Vi vill tacka alla som vi arbetat tillsammans med genom alla år.
Extra tack till granskarna av boken: Martin Jansson och Per Klingnäs.

James Bach, Heuristic Test Strategy Model, http://www.satisfice.com/tools/htsm.pdf

James Bach, Test Strategy (slides), http://www.satisfice.com/presentations/strategy.pdf

James Bach, Heuristic Test Planning: Context Model, http://www.satisfice.com/tools/satisfice-cm.pdf

Harry Collins, Tacit and Explicit Knowledge

Rikard Edgren, The Little Black Book on Test Design

Rikard Edgren, Henrik Emilsson och Martin Jansson, Kvalitetsegenskaper för programvara:
http://thetesteye.com/posters/TheTestEye_KvalitetsegenskaperForProgramvara.pdf

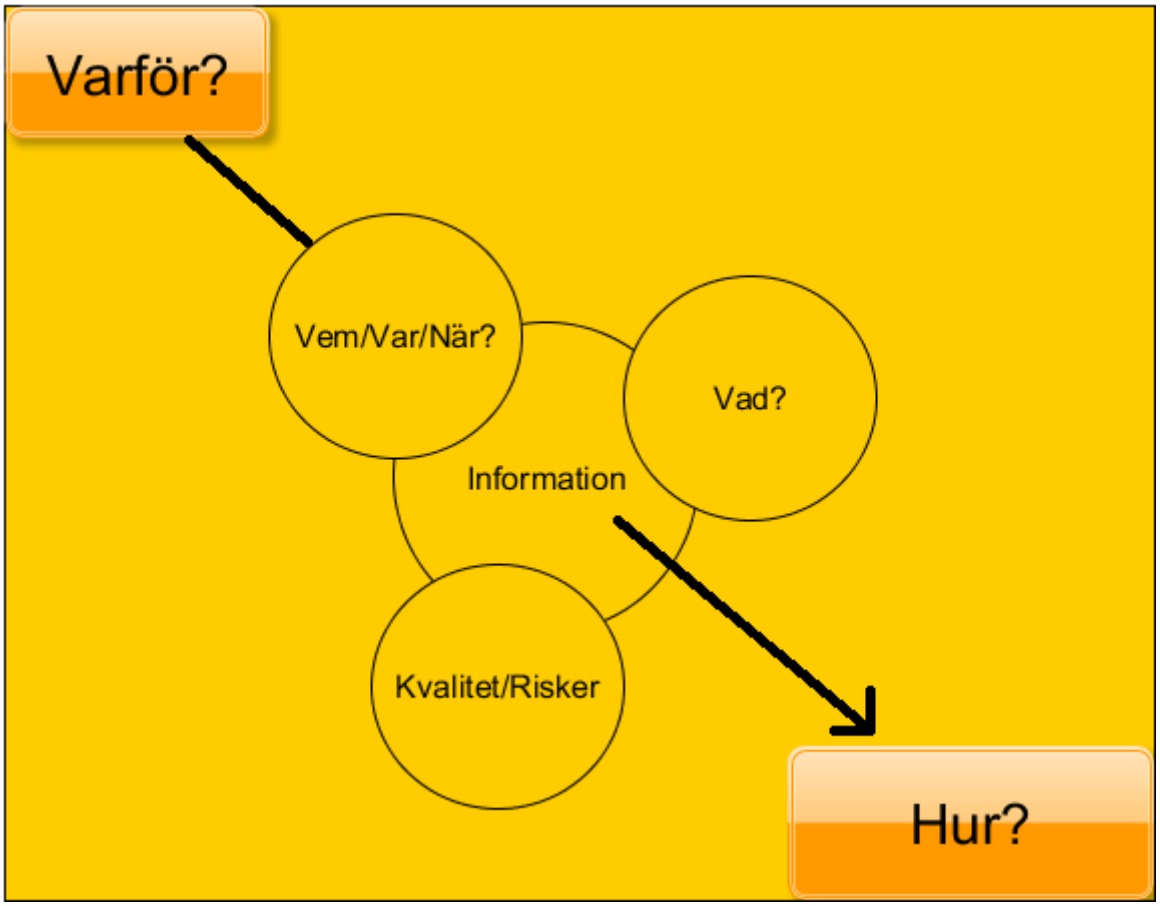
Rikard Edgren, Henrik Emilsson och Martin Jansson, 37 källor till testidéer:
http://thetesteye.com/posters/TheTestEye_KallorTillTestideer.pdf

IEEE 829 Test Plan Documentation, Approach section

Daniel Kahneman, Tänka, snabbt och långsamt

Cem Kaner, James Bach och Bret Pettichord, Lessons Learned in Software Testing

Cem Kaner och Rebecca Fiedler, Black Box Software Testing, Test Design,
http://www.testingeducation.org/BBST/testdesign/BBSTTestDesign2011pfinal.pdf


Varför?
Vem/Var/När?
Vad?
Information
Kvalitet/Risker
Hur?

Tycker du att läsarna är oväntat ointresserade av dina testresultat?
Har du undrat varför ingen tänkte på den testmetoden tidigare?
Är det svårt att veta var du ska starta planeringen av testinsatsen?

Då kan det här vara en bok för dig.

Målet är att läsaren ska lära sig hur man tar fram material för sin teststrategi, och hur man formulerar den så att många kan förstå, och tycka till.

Henrik och Rikard har arbetat med testning sedan slutet på 1990-talet. De hade vettiga teststrategier, men var svaga på att kommunicera dem. 2009 skrev Henrik en teststrategi Heureka! med utgångspunkt från Lessons Learned in Software Testing. År av diskussioner och praktisk användning har slutligen resulterat i tankar som nu nedtecknats.

Inkluderar
SFDIPOT (på svenska)
Kvalitetsegenskaper för programvara 1.1.1
37 källor till testidéer 1.0

This work is licensed under the Creative Commons Attribution-No Derivative License
V1.0, sjuttonde september 2014