0-fejl udvikling

Appendiks til bogen ’Struktureret Test’

1. Muligheder og forudsætninger

Det er muligt at udvikle og vedligeholde fejlfri systemer, hvis man er parat til at investere i det. De teknikker, der skal læres, og de regler, der skal følges, kræver ikke særlige mennesker, særlige omstændigheder eller særlige kunder. Det er primært en afvejning af investeringens størrelse i forhold til behovet.

2. Udgifter ved 0-fejl

Den første store udgift er tid. Der skal bruges så meget på hvert produkt, at det kan erklæres for fejlfrit. Hvor meget vil afhænge af, hvor tæt på målet man er i forvejen. Men der skal f.eks. bruges så meget tid på en analyserapport, at den kan erklæres både komplet og korrekt.

Den anden store udgift er værktøjer. Der er udgifter ved at evaluere, vælge, købe, implementere, uddanne i og supportere de nødvendige værktøjer. Men uden værktøjer går det ikke. Specielt i forvaltningssituationen. Værktøjer skal ses som en generel investering, fordi prisen kan løbe op i en størrelse, der kan blive svær at få plads til på et enkelt projekts budget. Og endnu sværere i forbindelse med en enkelt forvaltningsopgave.

Den tredje store udgift er uddannelse. Alle i en virksomhed skal uddannes i den model for test, man vælger at bruge. Det gælder også dem, der er dygtige til at teste i forvejen. Alle skal bruge samme metode, hvis alle skal udvikle fejlfrie systemer. Det er ikke nok at have et par stjerner, der kan det hele, men på hver sin måde.

Der er sat en pris på hvert af de midler til 0-fejl, der nævnes i afsnit 7. Den er ikke udtrykt i kroner og øre, men skulle nok give en fornemmelse af konsekvenserne ved at tage det i brug.

Referenceprojekt

For en del af de tiltag til 0-fejl der nævnes, er omkostningen afhængig af projektets størrelse. Samtidig med at omkostningen kun kan angives i tid. Der er derfor brug for et referenceprojekt. Det gør det også muligt at vurdere tiltagenes omkostninger indbyrdes.

Projektet er følgende: tre personer udfører 27 mandmåneders arbejde indenfor en periode på 12 kalendermåneder. Projektet omfatter alle udviklings- og testfaser, løser 270 Function Points, og bruger den moderate testmodel.

Der er kun angivet omkostningen for dette ene projekt. Det varierer meget ved større og mindre projekter, men giver i det mindste en ide om størrelsesordenen.

3. Indtægter ved 0-fejl

Indtægterne kommer for det første i sparede omkostninger. En fejl, der skal rettes efter at systemet er sat i drift, tæller godt i regnestykket. Sparede dagbøder og lign. går også ind i regnestykket. En endnu større besparelse opnås i systemforvaltningen. Når et system udvikles efter 0-fejl principper, bliver det ikke bare bedre testet, men også bedre dokumenteret og frem for alt enklere opbygget. Systemerne bliver mere overskuelige, og tilpasning, udbygning og sanering bliver betydelig billigere. Regn med at spare et sted mellem 30-50 % på omkostningerne til vedligeholdelse. Det kan bruges til at reducere mængden af ressourcer, der skal bruges på forvaltning.

Der er også indtægter i form af 'image' og 'kundetilfredshed', der udmønter sig i flere kontrakter. For interne IT-afdelinger drejer det sig om bedre relationer.

4. 0-fejl eller fejlfrie systemer

Der er forskel på '0-fejl' og 'fejlfrie' systemer. I sin rene udgave af 0-fejl handler det om ikke at begå fejl overhovedet. Hvis ambitionen kun er fejlfrie systemer, tillades en vis mængde fejl, blot de fjernes inden ibrugtagning. Med fejlfrit menes dermed fra systemet er sat i drift.

Der er således to faktorer, der øger testens effektivitet:

- Undgå at begå fejl
- Find flest mulige fejl i den tid der er til rådighed

Et eksempel på forskellen: Når programmører udfører fortolkende reviews på designdokumentationen forud for programmering, er det for at undgå at begå fejl i deres eget kommende arbejde. Når den færdige kode køres igennem Walk-Throughs, er det for at finde de fejl, der måtte være begået alligevel. Begge faktorer: *undgå* og *finde flere,* skal dyrkes for at få mest muligt ud af investeringen i test.

'Undgå at begå fejl'

Det er den mest effektive måde at producere fejlfri systemer på. Nogle vil hævde, at det er billigere at tillade en vis mængde fejl, blot de bliver fundet. Om de skal findes før eller efter ibrugtagning, er der også nogen, der er villige til at sætte op i regnestykker. Man skal så bare kunne styre, hvor mange fejl der begås.

Der findes ingen rigtig gode forsøg med at udvikle det samme produkt, men med forskellige krav til fejlprocenten. De få forsøg, der er gjort, er for gamle og for begrænsede. De tjener dog til at understrege det faktum, at man får det, man beder om. Forsøgene er udført ved at sætte et antal grupper til at udvikle samme program. Oplægget var ens for alle grupper, bortset fra en enkelt linie hvor der henholdsvis blev stillet krav om at:

Gruppe 1: programmet skulle være robust
Gruppe 2: udviklingstiden skulle være så kort som mulig
Gruppe 3: kildeteksten skulle være kort
Gruppe 4: det oversatte program skulle fylde mindst mulig
Gruppe 5: kildeteksten skulle være meget læsbar

Hver gruppe var bedst, med hensyn til det særlige krav. Uden at de vidste, at kravet kun var stillet til dem. Det viser, at der er større chance for at nå et bestemt resultat, hvis man beder om dét, end hvis man beder om noget andet.

Krav om fejlfrie systemer er ikke nok til at sikre det. En større effekt kommer, når man prøver at blive god til det. Når man træner en bestemt færdighed, bliver man støt bedre. Således er det også med udvikling af fejlfri systemer. Hvert nyt projekt kan bygge videre på de foregåendes erfaringer. Omkostningerne til udvikling og test falder, samtidig med at kvaliteten styres stadigt bedre.

Alternativet er at blive god til at finde og rette fejl. Man kan blive dygtig til det, og finde mange fejl pr. testtime eller pr. review. Det kan bruges som målestok for testens effektivitet. Men der mangler lidt perspektiv i den strategi. Effektiviteten stiger, når udviklerne begår flere fejl. Men det gør omkostningerne også.

Det grundlæggende kompromis
Den rette blanding af *undgå* og *finde flere* er princippet om, at rette fejl i den fase de begås. Det er et godt kompromis. Til gengæld skal man betragte fejl, der overlever faseskift, som en personlig tragedie. I hvert fald hvis man er tester. Det er det samme som to fejl. Først er den begået, og dernæst er det en fejl, at den ikke er fundet.

Kravet kan konkretiseres, ved at definere hvilke produkter i en fase der skal være fejlfrie. En designfase leverer skærmbilleder, databaser, moduldesign, design af dokumentation og design af testen. Designfasen kan defineres fejlfri, ved at kræve fejlfrihed i én, flere eller alle nævnte produkttyper.

Man kan sætte stadigt snævrere krav til det tidsrum, hvor en fejl må leve. Til sidst når man ned på, at hver en detalje skal være rigtig første gang. Det er det samme som at stille krav om det teoretiske 0-fejl forløb. Men lad det være målet i en proces, at nå dertil som et resultat af at øve sig på det.

Det er også sundt at huske, at resultatet af en fase er et fuldt og færdigt og fejlfrit system. Det er ikke et mellemresultat på vej til det 'rigtige'. Man kan ikke tillade sig at gå videre med en ukendt kvalitet, og håbe på at det endelige system bliver rigtigt. Det *er* det endelige system.

Der er en undtagelse, der sætter tolerancetærsklen for fejl lidt op. Principielt er det svært at overholde en tidsplan, hvis der findes mange fejl i sluttesten. Men en vis mængde konstruktionsfejl kan dog tillades. De er til at have med at gøre, selvom de hører til en foregående fase. De virkelige dræbere for en tidsplan er de fejl, der findes i sluttesten, og som er begået før konstruktionsfasen.

5. Testplanen til 0-fejl

Hvordan ser testplanen til et 0-fejl forløb ud? Den består af tre dele:

1. Forebyggelse (Prevention)
2. Opdagelse / Rettelse (Detection / Correction)
3. Tilbagemelding / Forbedring (Feedback / Improvement)

Punkt 2 og 3 består hver af to dele, men man skal planlægningsmæssigt se dem under et. Den første del er ligegyldig, hvis den anden ikke gennemføres, og den anden kan ikke udføres alene. 'Opdagelse' er intet værd uden 'Rettelse', og 'Tilbagemelding' er intet værd uden 'Forbedring'. 'Rettelse' kan ikke foretages uden 'Opdagelse' og 'Forbedring' forudsætter 'Tilbagemelding'.

De engelske betegnelser er de almindeligt brugte i testplaner, der bygger på retningslinierne i ISO 9000-3.

Punkt 1 sørger for, at projektet lander til tiden. Det sørger også for, at tiden og kronerne til test bliver en investering i stedet for en omkostning.

Punkt 2 er de traditionelle tests, men med vægten på statisk og tidlig test.

Punkt 3 sørger for, at det bliver stadigt billigere at blive færdige til tiden, og med aftalt indhold. Tilbagemeldingen sker både inden for den opgave, der udføres, og efter at projekter og forvaltningsopgaver afsluttes. Hver ny opgave står dermed på skuldrene af den forrige.

6. Proces / Produkt

Den ønskede kvalitet kan nås ad to veje. Den ene er at undersøge kvaliteten af produkterne, der kommer ud af fremstillingsprocessen. Den anden er at overvåge og styre fremstillingsprocessen.

Kvalitetsstyring af produktet

Tidligere tiders kvalitetskontrol lagde en kontrol ind i slutningen af fremstillingsforløbet. Princippet kan bruges, uanset om det er et stykke software eller et produkt på et samlebånd. For enden af samlebåndet sidder kvalitetskontrollen og kasserer de dårlige produkter (pass/fail). Men princippet er dyrt at bruge på software. Den høje stykpris på software gør ganske enkelt metoden uøkonomisk. Udviklingsomkostningerne er for store til at smide væk. 'Pass/fail' er bedst når fremstillingsomkostningerne er meget små. Derfor sker der ét af to med software. Enten repareres det færdige produkt, og sendes ud til kunden (det er det mest almindelige), eller også sendes det tilbage til analyse/design. Som et delvist 'fail'.

Der er i dag fuld erkendelse af, at en afsluttende 'pass/fail' kontrol er uøkonomisk for software. Uanset om beslutningen er reparation eller at dele af det skal gøres om fra start.

Det er bemærkelsesværdigt at industriel kvalitetskontrol også har forladt princippet. Kontrollen er rykket op ad samlebåndet. Dels på de dele der indgår i produktet, og dels på produktet i ufærdige udgaver. Ved at kassere tidligere, produceres der kun fejlfrie produkter til sidst. Princippet føres helt tilbage til indkøbet af hver del, med kontrol og inspektion på det tidligst mulige tidspunkt.

Når kontrollen bliver præventiv, og integreres med fremstillingen, bliver det kvalitets-*styring* i stedet for *-kontrol*.

Metoden er udviklet så langt, at tilbagemelding og forbedring sker løbende. Når der findes en fejl, stopper produktionen, og den ansvarlige kvalitetsgruppe samles på stedet. Årsagen skal findes med det samme, for at afgøre om det drejer sig om en enkeltstående svipser, eller om der er noget mere galt. Hvis det sidste er tilfældet kan hele produktionen først genoptages, når årsagen er fundet og fjernet. Princippet er helt klart: Hvis noget er forkert, må der ikke bruges en krone mere på det. Både når det drejer sig om et enkelt produkt eller en del af produktionsforløbet.

Princippet om kvalitetsstyring af produktet anvendes til systemudvikling ved at sætte detaljerede og præcise krav til strukturen og omfanget af udviklingsdokumentationen. Og præcise krav til udseendet af dens indhold. Ved hver milepæl testes alle produkter omhyggeligt, og de dårlige dele kasseres.

Kvalitetsstyring af processen

Styring af produktet kan forhindre at fejl får lov at leve videre, og det kan godt stå alene som kvalitetsstyring. Men på et tidspunkt bliver der også økonomi i at styre processen, altså måden vi arbejder på. Hvis man gerne vil undgå at kassere produkter. Efterhånden som kontrollen rykker op ad samlebåndet, åbnes der jo mulighed for at vurdere og styre hvordan det *vil blive*. Og ikke bare hvordan det *er* skruet sammen. Man kan f.eks. stille krav, til den måde kravspecifikationen fremstilles på.

Det kan diskuteres længe, hvor nødvendigt det er at styre processen. Man kan mene, at rigtige resultater kan nås alene ved at stille krav til produktet. Således har det næsten traditionelt været ved fremkomsten af hver ny systemudviklingsmetodik. Men uanset

hvor detaljerede krav, der stilles til færdige kravspecifikationer, klassemodeller eller Use Cases, så bliver styring af processen på et eller andet tidspunkt interessant.

Afvejning af produkt/proces
Afvejningen af produkt og proces er et spørgsmål om økonomi. Umiddelbart lyder styring af proces jo særdeles fornuftig, fordi fejlene kan findes, før de begås. Men nogle produkter er så billige at producere, at det bedre kan betale sig at kassere dem efter fremstilling.

'Proces' er desuden mange ting. Man kan lægge vægten på arbejdsgange, procedurer eller noget helt andet.

Forudsat en tidlig test, kan overvejelserne vises på en testplan, og andre kan vurdere, om det er de rigtige tiltag, der investeres i.

Test af produkt har sin fordel i, at man kan se hvad der testes. Problemet ligger i kassationen. Test af proces har sin fordel i at forhindre fejl, men den er sværere at teste.

7. Midler til 0-fejl

De enkelte tiltag der gennemgås i det følgende, bidrager til at *undgå* fejl, *finde flere* fejl, eller begge dele. Der er mange flere tiltag der er nyttige, men som ikke nævnes. De er nyttige, når man ser på test som en arbejdsopgave, men bidrager ikke direkte til at *undgå* eller *finde flere* fejl.

Man kan ikke sikre 0-fejl ved at udnævne et af tiltagene til en mirakelmedicin. For dem der mener, at 'Prototyping' klarer alt, 'Reviews er det eneste saliggørende middel', 'brug OO og du har ikke brug for test' eller noget lignende, er dette en lejlighed til at spare tid. De kan blot springe kapitlet over. De vil også få god brug for tiden til at finde og rette fejl.

Betragt derimod listen som nogle muligheder der kan kombineres. Valget er afhængigt af tiden der er til rådighed, opgavens vigtighed, den forventede nytte kontra udgiften, og de praktiske muligheder der er for at sætte dem i værk.

I det følgende gennemgås:

1. Offensiv strategi
2. Verifikation og validering parallelt
3. Testenheden fastlægges før/under analysefasen
4. Design med test i tankerne
5. Alle farver test
6. Testleder i projektet
7. Testere i projektet
8. Testdesign før systemdesign
9. Testdata før kodning
10. Test af test
11. Seriøs udviklingsdokumentation
12. Testspecifikation udvikles seriøst
13. Box Structured Design
14. Automatisk regressionstest
15. Holdninger
16. Standarder
17. Dækningskontrol

18. Refactoring frem for reparation
19. Testplanlægningsseminar
20. Erfaringsopsamling (Knowledge Management)
21. Involvere interessenter
22. Halsted & McCabe
23. Prototyper
24. Tænke-Højt-Test
25. Structure Correlation
26. Køb testen
27. Testorganisation
28. Fejlårsagsanalyse (Defect Causal Analysis)
29. Integreret sporbarhed
30. Defensive Programming
31. Test som mikrodesign
32. Pair Testing
33. Procesforbedring (TMM)
34. Paralleltest
35. Betatest

7. 1 Offensiv strategi

Ved at starte gennemførelsen af test samtidig med udvikling satses der kraftigt på at undgå fejl. Man sikrer sig et ansvar for testen, og man sikrer sig, at der er ressourcer til at gennemføre den. Ellers vil det hurtigt blive synligt. Man kan ikke nøjes med at køre en offensiv strategi på papiret. Reviews vil hurtigt komme på banen, hånd i hånd med tidlige prototyper eller andre former for modeller.

Problemerne i en offensiv strategi er i virkeligheden fordele. Det største problem er jo, at der ikke er et system at teste. Og det er jo netop det, det drejer sig om: at teste før systemet er der. Kravspecifikationen bruges som beslutnings- og inspektionsgrundlag. Systemet modelleres og testes. En offensiv strategi kan også indebære at programmører fortolker designresultatet med brugere som tilhørere.

Pris?

Primært ressourcer. Regn som et minimum med at 25 % af de samlede ressourcer i foranalyse og analysefasen bruges til test i en offensiv strategi.

Hvis der er brug for prototyper, og man ønsker at bogføre det under test, skal det lægges til. Regn med 30 % til test i det tilfælde.

7.2 Verifikation og validering parallelt

Kombinationen af at køre V&V parallelt sørger for at der anlægges to syn samtidig. Tag eksemplet med et sæt færdige Use Cases. Brugerne validerer om de er korrekte, og designerne verificerer, at de kan omsætte dem til et design.

I samspillet med en offensiv strategi er det yderligere effektivt. Verifikationen giver en solid forebyggelse. Antallet af fejl i næste fase kan bringes mod 0.

Pris?

Flere reviews i de tidlige faser, og større krav til dokumentationen. Når V&V køres parallelt bliver der også brug for mere koordination. Men der er ikke brug for ekstra værktøjer eller mere uddannelse.

Prisen på det aktive kontra det passive kommer meget godt frem her. Hvis man kører verifikation af et færdigt designresultat op imod Use Cases, er der brugt en passiv strategi. Hvis der i stedet bruges en måned i referenceprojektet, på at lade designerne stave sig vej igennem Use Cases i form af fortolkende reviews eller inspektioner, er mulighederne for 0-fejl fremmet betydeligt.

Hvis analyse og design udføres af de samme personer, skal verifikationen udføres af eksterne testere. Der er stadig brug for en måned. Det er ikke gjort med nogle reviews på et par timer hver. Det giver ikke skygge af chance for et 0-fejl forløb.

7.3 Testenheden fastlægges før eller under analysen

Lad os tage prisen først: 0. Det koster højst en times arbejde på et opstartseminar, der måske skal holdes alligevel. Hvis der holdes et testplanlægningsseminar, er emnet selvskrevet. Uanset seminarerne skal testenheden vælges på et eller andet tidspunkt. Det handler blot om at fremrykke tidspunktet.

Gevinsten er færre fejl. Selve det forhold at man bruger en enhed på testplanen, medvirker til at finde flere fejl. Men skal den fastlægges før eller under analysefasen? Valget går klart ind og styrer forløbet. Selv om valget bliver så uopfindsomt som 'funktioner', hjælper det med til at få listen over funktioner udarbejdet klart og tydeligt. Der bliver brug for, at aftale konkret hvad der er funktioner, både mellem udviklere og testere og mellem testere og de øvrige interessenter.

Pris?

Som sagt, gratis. Det koster ikke mere tid at fastlægge den tidligere. Det er kun i tilfælde af, at det ikke havde været planlagt at gøre overhovedet, at det koster noget tid. Men det havde med stor sandsynlighed kostet fejl i produktionen, på grund af oversete fejl i den mangelfulde testdækning.

7.4 Design med test i tankerne

Effekten er, at antallet af programmeringsfejl falder dramatisk. Regn med at kunne fjerne 90 %.

De vigtigste elementer i 'Testbarhed' er nævnt i bogens kapitel om systemegenskaber. De handler om 'Design for Testability'.

Men det er klart et præventivt middel vi har fat i. Mange anser det for at være det vigtigste i forbindelse med programkvalitet. Det er desuden et bidrag til at finde flere fejl, fordi det afslører nogle forhold, der rækker tilbage i udviklingsforløbet.

Pris?

I vort referenceprojekt skal man forlænge designfasen med 4-6 uger. Det skal afvejes med besparelsen på 90 % færre programmeringsfejl.

7.5 Alle farver test

Brug bevidst alle farver test, både 'White-, Black- og Grey-Box'. Det finder flere fejl. Hvis man kun læner sig op ad en af dem, fås en ret ineffektiv test. Det bliver til sidst gniderier ude i hjørnerne. Der er for lille effekt i at motionere de sidste teoretiske veje i et modul, blot for at hæve dens dækningsgrad fra 97,8 til 98,7. Hvis der i stedet kan sættes ind på en af de andre farver. Man kan heller ikke regne med at selv den bedst tilrettelagte brugertest, altså Black-Box, kan erstatte en White-Box fuldstændig. Det vil give en dårlig brugertest, hvis man springer fra programmering og over i Black-Box.

Pris?
Gratis.

7.6 Testleder i projektet

Ved at give en person ansvaret for testen delegeres en del af det samlede opgaveansvar. Ansvaret for testen kan også placeres i en særlig organisatorisk enhed. Den får så typisk det generelle ansvar for test og kvalitetsstyring i virksomheden.

Der er både mulighed for at *undgå* og *finde flere* fejl. Det kommer an på, hvilken strategi, der er valgt. Disse 0-fejl tiltag spiller sammen på en måde, der påvirker omfang og effekt. Hvis der er valgt en offensiv strategi, har den testansvarlige mulighed for at sætte tiltag i gang på begge fronter. Hvis der er valgt en traditionel strategi, er der kun mulighed for at *finde flere* fejl.

Det separate testansvar ses ofte brugt på 'Brugertest' og/eller 'Systemtest'. Det er også glimrende for de faser, men hvorfor ikke udvide ansvaret. Grunden til at begrænse det til f.eks. brugertesten, er at den person der får ansvaret, udvælges på faglige kriterier. Den testansvarlige får kun et ansvar, der passer med vedkommendes faglige indsigt. Det behøver ikke være en reel begrænsning. Der er intet i vejen for at have det samlede test-ansvar, selvom den faglige indsigt kun ligger på en del af testopgaverne. Det er nok lidt tradition, der gør sig gældende. Indenfor almindelig organisationsteori lader man sig ikke begrænse på den måde. Men mange udviklere stejler, når en bruger får ansvaret, ikke alene for brugertesten, men også for White-Box test i modultesten. "Sådan noget har de ikke forstand på".

Pris?
Beskeden. Der er måske et kursus, der skal tages, suppleret med en bog der skal læses. Men det primære er at finde nogen, der har lysten til at udføre arbejdet med lyst og grundighed. Det er vigtigere at komme i gang med jobbet, end at betragte det som en videnskab.

7.7 Testere i projektet

Effekten af at bruge separate testere er som de selv eller en testansvarlig ønsker. Pointen er jo, at der sættes nogle hoveder af alene til at finde og undgå fejl. Det er en meget behagelig situation, og effekten kan være kolossal.

Traditionelt har separate testere mest været brugt til at finde fejl. Kun i begrænset omfang har de været brugt til at forhindre fejl. Det kan jo undre, men det er en udvikling, de fleste brancher skal igennem. I IT-branchen er vi kun nået til systematisk præventiv test

på risikofyldte projekter. Men der er økonomi i at gøre det på alle typer projekter og applikationer.

Der er en parallel til det foregående punkt, med hensyn til at tænke i baner af hvad testerne har af faglig kompetence, men det behøver ikke være sådan. En brugertester kan organisere reviews inden for områder, der rækker langt ud over, hvad den faglige viden tilskriver.

Pris?
Her er der ingen parallel til det foregående punkt. Fordi her er udgiften ikke beskeden. Den tæller med mandtid, og det løber jo hurtigt op. Hvis udgiften lægges på toppen af de øvrige udviklingsomkostninger, tæller de direkte. Hvis man i forvejen har afsat en del af udviklernes tid til test, og kan forsvare at overføre den på nogle separate hoveder, kan den samlede udviklingstid reduceres med den del. Desværre er det ikke altid tilfældet. Det er ikke ualmindeligt, at de tidsplaner, der lægges for en udviklingsopgave, regner med at forløbet bliver ret fejlfrit. At der ikke er kalkuleret voldsomt meget med tid til at få det fejlfrit. Så er der heller ikke noget at overføre.

Men hvad skal man sige til det? 0-fejl udvikling er mulig, men det kommer nu engang ikke af sig selv. Nogen skal få det til at ske. Og det koster tid.

Man kan omvendt gøre sig nogle overvejelser om, hvor meget tid der maksimalt kan sættes af til testere. Svaret ligger mellem 25-45 %.

Et godt råd er altså at lade mindst 25 % af ressourcerne være øremærket til at sørge for, at det bliver et fejlfrit produkt. Og brug højst 45 %.

Disse tal er anderledes end de tilsvarende for hardwareudvikling. I den branche er der mulig nytteværdi ved at bruge helt op til 80 % af de samlede ressourcer på test. Det levner kun 20 % til udvikling, og giver alligevel et bedre produkt. Der spiller selvfølgelig nogle forhold ind omkring omkostninger på at rette hardware kontra software, som betinger forskellen. Kravet til hardwares korrekthed er større, fordi omkostningerne ved at tilbagekalde og rette er større. Men softwarebranchen burde som helhed præstere bedre produkter, også selvom den ikke skal tilbagekalde fra alle egne af jorden, eller skal yde erstatning på hele købssummen.

7.8 Testdesign før systemdesign

Den praktiske del af dette tiltag er at skrive en testaktivitet for hver forekomst af den valgte testenhed. De skrives efter analyse og før design. Det har effekt på to måder.

Mængden af fejl i designfasen reduceres. Det er altså et bidrag til at *undgå* fejl. Tiltaget er et 'must' hvis målet er 0-fejl i designfasen. Alternativet er at skrive testaktiviteter efter design af systemet. Så kan man lige så godt vente til efter konstruktionen, modultesten, og integrationstesten. Det giver ingen større forskel, hverken med at *undgå* eller *finde flere* fejl.

Der kan desuden findes fejl i analyseresultatet. Hvis der har været et 0-fejl forløb frem til dette tidspunkt, er der ikke noget at finde. I modsat fald er her et middel til i det mindste at få fejlfrie systemer ved at finde dem. Det er (næsten) en mulighed for at finde fejl, i den fase de er begået. Reelt er analysen afsluttet, og designet skal begynde. Men det er så tæt på analysen, at man har klaret at overholde princippet.

Vi kan også pege på, hvad det erfaringsmæssigt er for fejl, der findes i analyseresultatet: uklarheder. En del af analysen kan være opfattet forskelligt af forskellige personer. Hver for sig har de en klar opfattelse af, hvordan det skal forstås, den er blot for-

skellig. En klassisk fejl der har bragt mange tidsplaner i skammekrogen. Det koster 5 min. at rette i analysen, men to dage i systemtesten. Den type fejl kan findes ved at udarbejde testdesign før design af systemet.

Pris?
Beskeden, hvis man i forvejen har tænkt sig at designe testen. Så er det kun et spørgsmål om at rykke aktiviteten frem i forløbet. Hvis der ikke er planlagt med at designe testen, koster det noget at begynde på det.

7.9 Testdata før kodning

Der er tre muligheder for at skrive testdata, nemlig før, samtidig med eller efter at det produkt, der skal testes, bliver produceret. Når målet er fejlfrie systemer er sagen klar. Testdata skrives før.

Før analysen begynder, udarbejdes der altså scenarier til at teste den med. Før designet begynder, udarbejdes der testforløb omkring brugervenlighed, robusthed etc. Før programmerne skrives, udfærdiges testdata der kan bruges til Walk-Throughs.

Det er et middel til at undgå fejl. Forudsat at nogle betingelser er opfyldt. For det første den helt indlysende, at dem der skriver produktet, der skal testes, får disse testdata til deres rådighed. Så de kan bruge dem som facitliste. Det leder til næste forudsætning, at testdata skal være korrekte. F.eks. skal testdata til test af programmer, skrives på grundlag af de samme specifikationer, som udviklerne bruger til at skrive programmer. Det giver to fejlmuligheder. Specifikationen kan være forkert, og dermed bliver der skrevet forkerte testdata. Eller testdata skrives blot forkert. Hvis det er på grund af fejl i specifikationerne, ligger årsagen til det tidligere i forløbet. Så er det kun godt at de bliver fundet. Hvis de kun stammer fra at skrive testdata, er det nemt at rette.

I begge tilfælde kan testdata reviewes af brugere, og i begge tilfælde er der givet et væsentligt bidrag til at undgå fejl i de efterfølgende faser. Bemærk at brugere kan reviewe testdata, men ikke programmer.

Hvis systematikken bruges fra start til mål med at skrive facit før udvikling, er vi igen endt i 0-fejl udvikling. Sådan som det foregår i højrisikoprojekter. Det er ikke nødvendigvis vores ambition, men det gør ikke noget, at forløbet udbygges løbende til at blive det.

Pris?
Der skal bruges ekstra tid på at skrive facitlisten. Det er billigere at gøre det efter, at produktet der skal testes, er fremstillet. Fordi man kan spørge udviklerne, hvordan systemet skal virke. Det er jo også meget godt, bortset fra de tilfælde hvor deres opfattelse er forkert.

Den ekstra tid der bruges på at skrive testdata, skal afvejes med deres værdi som selvstændig facitliste.

7.10 Test af Test

Det er et middel til at finde flere fejl. Igen kan man ved at definere nogle passende forudsætninger, også argumentere for, at det hjælper med til at undgå fejl. Men det er ikke nødvendigt at presse teoretiske fordele ud af det. Det er under alle omstændigheder et så godt middel til at checke testforløbet, at det er investeringen værd.

Pris?
Nogle timers indsats. Enten fordelt over flere omgange, eller gennemført samlet efter design af testen. Mange spørgsmål kan besvares allerede i specifikation af testen og analyse af testen, og det er fint at gøre det. Prisen er så beskeden, at man kan forsvare at køre 'Test af Test' igennem flere gange.

7.11 Seriøs udviklingsdokumentation

Det kan synes overflødigt at pege på værdien af en seriøs udviklingsdokumentation. Det er den vel altid? Eller stilles der særlige krav til den, hvis vi ønsker 0-fejl udvikling?

Svaret er i høj grad et 'ja'. Der stilles særlige krav til godkendelsen, godkenderne og dokumentationen.

Tiltaget gælder især kravspecifikationen med tilhørende Use Cases. De skal være fejlfrie, og målet er at undgå fejl i det resterende projektforløb.

Der er mange eksempler på udmærkede dokumentationer af f.eks. analysefaser. Som også er blevet formelt godkendt af interessenterne. Men der stilles andre krav til godkendelsen, når vi ikke accepterer fejl. Så er det ikke nok med en godkendelse af analysefasens produkter. Så er der brug for en godkendelse af, at det er det færdige system, vi står med.

Dermed stiller det særlige krav til godkenderne. De skal have kompetence til at godkende det færdige system.

Endelig stiller det særlige krav til selve dokumentationen. Den skal være testbar. Udarbejdet således at interessenterne kan læse, forstå og godkende den som værende komplet og korrekt.

Hvis en specifikation indeholder diagrammer eller modeller, er det da fint. Det kan fx. være datamodeller eller OO-modeller. Men de er intet værd i denne sammenhæng, hvis de ikke kan forstås hundrede procent af interessenterne. I første række køber og bruger. Ellers kan de kun bruges som intern udviklingsdokumentation. Det er OK at undervise bruger og køber i modellering. Bare det gøres med et resultat som giver reel mulighed for at kunne erklære modellerne for fejlfri.

Pris?
Prissedlen er ikke uinteressant. Der skal regnes med et måneds ekstra arbejde i referenceprojektet.

7.12 Testspecifikation udvikles seriøst

Ingen regel uden undtagelse. De tiltag der nævnes i dette kapitel, skal kun omfatte dem, der direkte medvirker til 0-fejl. Det er ikke nok at gøre det indirekte. Men testspecifikationen fungerer som foranalyse på de samlede testaktiviteter. Den specificerer og dokumenterer, hvordan 0-fejl nås. Og at det overhovedet er målet. Derfor får tiltaget lov til at komme ind i varmen.

Effekten er en strammere styring. Specifikationen dokumenterer hvem der tester, og detaljerer seriøst hvornår og hvordan de gør det.

Lad os tage et eksempel. Afslutningskriterierne for testen er en del af specifikationen. Man kan antage at det er nok at få dem udarbejdet og dokumenteret. Men en seriøs testspecifikation vil detaljere og vise dem på en måde, der gør det muligt for andre at veri-

ficere dem som korrekte. De vil blive lagt til grund for styringen og blive brugt til at måle testens fremdrift.

Pris?
Hvis vi forudsætter at testen under alle omstændigheder specificeres, koster det kun tiden til at skrive dokumentet. Da der kun er tale om et dokument på nogle få sider, er udgiften tilsvarende nogle få timer. Hvis testen ikke specificeres i forvejen er prisen en anden. I referenceprojektet ca. en uge.

7.13 Box Structured Design

Formålet er at *undgå* fejl i design-fasen.

Systemet designes som en række 'bokse'. For hver 'boks' beskrives dens funktion sammen med de data den bruger. Moduldiagrammet viser dermed samtlige funktioner og data i systemet. Hver 'boks' dækker en veldefineret del af kravspecifikationen, og hvert krav dækkes af en eller flere hele 'bokse'. Den efterfølgende programmering er dermed også lagt i faste rammer. Det færdige fysiske system kan følges baglæns til design og analyse.

En 'boks' adskiller sig fra objekter ved primært at vise funktion. Der er heller ikke de samme formelle krav til beskrivelser.

Der kan også regnes med at finde evt. analysefejl. Med mindre analysen er gennemført fejlfrit. Men så kan man glæde sig over at få det bekræftet. 'Box Structured Design' fungerer nemlig som Verifikation af analysen. Eventuelle mangler i designet som følge af mangler i analysen, ses tydeligt. Det er ganske enkelt ikke muligt at få et komplet design.

Der undgås fejl i konstruktionen. Designet vil være komplet og konsistent. Det nyder især kodningen godt af. Programmeringen bliver hurtigere og med større kvalitet. Der gælder ikke det samme for den øvrige konstruktion, f.eks. skærmbilleder, databaseviews og dokumentation. Det kræver andre tiltag, f.eks. prototyper.

Det giver også forvaltningsvenlig kode, og dermed bedre muligheder for at *undgå* og *finde flere* fejl i forvaltningen.

Pris?
Vi må hellere regne med Worst Case, og det vil sige en situation hvor der ikke bruges formelle design-metodikker i forvejen. Så vil det koste 4-6 ugers ekstra arbejde i referenceprojektet.

7.14 Automatisk regressionstest

Regressionstesten finder en særlig slags fejl, nemlig dem der opstår efter, at andre er rettet. Til det formål er et automatisk værktøj afgørende for produktiviteten. Det er ikke muligt at foretage en manuel regressionstest, og da slet ikke et større antal gange.

Hvis man kører et totalt 0-fejl forløb, er der ingen rettelser og værktøjet er overflødigt. Men hvis det er et forløb med et vist kompromis, er der brug for det. Det er et klassisk passivt testværktøj. Der er ingen støtte til at undgå fejl.

I forvaltningssituationen er det bestemmende for hvor meget af det urørte system der i praksis kan testes efter et indgreb.

Pris?
Som det er tilfældet med mange andre værktøjer, kommer de i mange ambitionsniveauer og prislag. Det afhænger af antallet af piber og trommer som værktøjet er udstyret med.

7.15 Holdninger

Mange mener at holdninger er det vigtigste middel til at opnå resultater. At menneskets potentiale er umådeligt, og at intet mål er uopnåeligt. Heller ikke 0-fejl udvikling. Således at hvis alle der er involveret i en opgave, har en holdning om ikke at begå fejl, vil det blive sådan.

Selv med holdningerne på plads kan der nu nok være brug for nogle af de andre tiltag til at bakke dem op med. Om ikke andet så for at demonstrere sine holdningers effekt.

Men det er da rigtigt, at holdninger har stor betydning. Når man har arbejdet med praktisk systemudvikling og har oplevet mange forskellige menneskers indsats og attituder, ja kort sagt 'forskellige holdninger', er der ingen tvivl om betydningen.

Man kan blive nedslået, når man møder en ligegyldig holdning. Hvor der er gjort meget lidt eller intet for at checke et produkt, før det f.eks. sendes til brugertest. Enten løber de an på heldet, eller også er de måske bare ligeglade.

Omvendt bliver man opstemt af andres positive holdninger til kvalitet. Når nogen i gruppen markedsfører ideer og holdninger til at producere kvalitet. Så bliver der plads til at dele holdninger som:

- Vi vil ikke begå fejl.
- Vi vil finde de fejl der alligevel måtte smutte igennem

og formulere dem i skrift og tale.

Ordet VI er brugt fordi fælles holdninger er en nødvendighed. Alle skal føle et ansvar, for at det lykkes.

Pris?
Gratis. Kvalitet er i øvrigt gratis. Men det er måske bare en holdning.

7.16 Standarder

Standarder er både med til at *undgå* og *finde flere* fejl. Pr. definition er det en fejl at afvige fra en vedtagen standard. Så på den måde kan man undgå en del fejl ved at følge selv de tåbeligste standarder. Men en god standard sætter en nyttig norm, og er et udtryk for en virksomheds samlede og ofte dyrekøbte erfaring. Den sikrer at hvert nyt projekt står på skuldrende af det forrige.

Der kan vælges mellem et uendeligt antal standarder på et uendeligt antal områder. Der bør med omhu vælges dem, der er mest nyttige, når alle følger dem.

Standarder for udvikling gør livet lettere for testere. Der bliver nemlig færre fejl at finde. Standarder for test gør også livet lettere. Fejlene kan nemlig findes hurtigere.

Checklister fungerer som bærere af standarder. De er en god fysisk implementering. De spiller en central rolle, når der satses på kontinuerlig forbedring.

Pris?

Gratis, forudsat at der er enighed om indholdet. Hvis der er uenighed, er tiltaget ligegyldigt. Uanset hvor mange penge der puttes i det. Checklister skal udvikle sig over tiden. Det gør dem bedre, plus at omkostningerne kan begrænses til tiden det tager at registrere en erfaring, der allerede er betalt.

7.17 Dækningskontrol

Dækningskontrol er en håndfast måde at finde fejl på. Ved objektivt at måle hvor meget der er testet.

Analyse- og designresultatet dækkes ved hjælp af testaktiviteter. Kontrollér dækningen ved at sammenligne testplanens aktiviteter med systemets indhold. Konstruktionsresultatet dækkes ved hjælp af testcases.

Den maskinelle dækningskontrol er på kode. Den eneste farbare vej er at købe et værktøj til at gøre det med. Det kan så automatisk kontrollere de forskellige typer af dækning. Det kan være kontrol på instruksniveau (Statement Coverage), på valg (Decision Coverage/Branch Coverage) eller på stier (Path eller Flow Coverage).

Den maskinelle beregning af dækningen bidrager til den løbende opfølgning og dokumentation af testens fremdrift.

Pris?
Samme kommentar som for værktøjerne til regressionstest.

7.18 Refactoring frem for reparation

Hvem vil ikke gerne have en ny og urepareret bil fra robotternes hånd, frem for en der har været tabt på gulvet et par gange. Det kan da godt være, at den er repareret så der ikke er nogen forskel, men alligevel.

Det samme gælder når bilen har været på værksted. Det er måske billigere at rette det bøjede ud, i stedet for at sætte helt nye reservedele i. Men hvor længe har man glæde af det.

Der ligger mere end et princip i dette tiltag. Der er en reel effekt med hensyn til at undgå fejl. Når et produkt har været repareret meget, stiger sandsynligheden for, at der opstår afledte fejl senere. Det gælder alle produkter i et udviklingsforløb. Det er måske mest tydeligt for kodens vedkommende, men det gælder for eksempel også Use Cases.

Princippet har vundet indpas i forvaltningsbevidste organisationer. Fejl har det med at klumpe i bestemte moduler, hvorimod andre dele af systemet bare kører som en drøm fra første dag. Dorothy Graham: "Defects are social creatures".
Tag og udskift de mandagsmoduler.

Pris?
Produktet skal skrottes, hvis det koster mere at lade det leve, end at skrive det om.

7.19 Testplanlægningsseminar

Effekten er både at *undgå* og *finde flere* fejl. Det er et middel til at træffe de nødvendige aftaler. Selv enmandsprojekter bør bruge det, og det er måske netop dem, der lettest overser midlet. En mindre forvaltningsopgave, der kun udføres af én person, kan sagtens testplanlægges sammen med en bruger, en opdragsgiver eller andre interessenter.

Det er ikke nødvendigt at kalde det et planlægningsseminar, bare indholdet er det samme.

Pris?
For større seancer er det et beløb, der kan mærkes. Et seminar på to dage med fem personer skal reelt bevilges.

På referenceprojektet vil et seminar på to dage være passende.

7.20 Erfaringsopsamling (Knowledge Management)

Der kan begås færre fejl ved at studere, hvilke der er begået i andre projekter. Der kan også *findes flere* fejl, ved at bruge de erfaringer andre har gjort, og gentage og udbygge de teknikker der har givet størst afkast, i forhold til investeringen.

Der kan siges meget om, at udviklingsarbejde er unikt. At den opgave der løses, aldrig har været løst før, og aldrig skal løses igen. Derfor er andres erfaring billet til det tog, der kørte i går. På den anden side har dele af ethvert projekt, været udført mange gange før. Så på udvalgte områder kan andres erfaringer bruges. Det kan være omkring test af databaser, testplanlægning eller samarbejdet med brugerne. Fordi brugerne måske er de samme, testplanerne skal måske godkendes af de samme mennesker, og databasen ligner den, der blev brugt i går og skal bruges i morgen.

Pris?
Deltagelse i projektevalueringer, og studier af evalueringsrapporter. De fleste af dem indeholder et afsnit om test.

Investeringen kan være generel ved at orientere sig om alle evalueringer løbende, og høre om alle erfaringer generelt. Den kan gøres mere målrettet ved at studere afsnittene om test i de seneste fem projekter, på det tidspunkt man selv skal i gang.

7.21 Involvere interessenter

Dette nævnes kun for at gøre listen komplet. Det er et grundlæggende middel, som må formodes kendt af alle:

- Man kan reducere mængden af fejl, der begås, ved at involvere dem, der afgør, om noget er en fejl, i selve arbejdet.

- Man kan finde flere fejl ved at bruge interessenterne som testere. Brugere til brugertest, forvaltere til robusthed og forvaltningsvenlighed, testere til testbarhed, etc.

Pris?
Det er gratis. Medmindre man skal betale for at få det gjort, qua interne projektaftaler og intern kontering. Eller når brugerne er eksterne, og der skal betales for deres tid. Men ellers er det et spørgsmål om at bruge sin fantasi. Find på måder til at involvere dem på afgørende tidspunkter, uden at gøre det via kontrakter. I øvrigt er muligheden der jo for at betale timeprisen. Der er ingen, der siger, at det skal være helt gratis.

7.22 Halstead & McCabe

Lad os tage de to herrer som fællesnævnere for de tiltag der går ud på, at undersøge hvilke dele af koden der er de mest komplekse. Ud fra en idé om at de måske så ikke er værd at teste. Effekten er at undgå fejl, ved at skrive den mest komplekse kode om, uden at begynde at teste den overhovedet. Det er en test af kvalitet.

Lidt mindre radikalt kan man nøjes med at ofre særlig opmærksomhed på den komplekse kode. Der findes i så fald flere fejl pr. testtime.

Flere normer omkring kode følger automatisk i kølvandet. I første omgang modulstørrelsen. For at få et acceptabelt resultat i en måling, er små moduler en væsentlig faktor. Men alle øvrige forhold omkring kvalitet af den fysiske kode såsom læsbarhed og vedligeholdelsesvenlighed, indgår også i målingen.

Pris?

Værktøjets. Der skal foretages en investering på virksomhedsniveau, som først kommer igen, efterhånden som værktøjet bruges. For det enkelte projekt er det en ren gevinst, fordi brugen af værktøjet tager sekunder.

7.23 Prototyper

Prototyper er både et middel til at *undgå* og *finde flere* fejl. I første omgang bruges en prototype til at finde fejl i den fase, som den dokumenterer. Efter en analyse bygges f.eks. en prototype, der kan testes og debugges. Gør den mere aktiv ved at bede designere om at tolke prototypen så langt ned i detaljen, at de kan beskrive det kommende system. Det forhindrer designfejl.

Der er grund til at være realistisk omkring nytten. Prøv at tage listen over egenskaber og check hvilke, der er tale om. Det er primært komplethed, nøjagtighed og brugervenlighed. For de øvrige egenskabers vedkommende er der mindre grader af effekt.

Prototyper er et eksempel på, at der skal mere end ét tiltag til.

Der opstår en særlig situation når man ikke er i stand til at bygge en prototype efter analysens afslutning. Når udviklerne siger, at det ikke er muligt, fordi de ikke har tilstrækkelig viden om indholdet. Hvad skal man så gøre? Faktisk er det en glimrende test. Vi har fået et testresultat, der siger, at analysen ikke er god nok.

Pris?

Der er en værktøjspris og en omkostning i timer. Værktøjets pris er gratis når muligheden er til stede på udviklingsplatformen, eller hvis værktøjet er købt i forvejen. Det er beskedent, selv hvis det skal købes først. For nogle få tusinde kroner kan der købes værktøjer, der er skræddersyet til formålet. Det er byggetiden, der reelt skal overvejes.

Prøv at bruge prototypen til test i stedet for udvikling. Lad en tester bygge en prototype på grundlag af specifikationen. Testeren kan være en IT-professionel eller en bruger. Der er fordele og ulemper ved begge muligheder. Det er i begge tilfælde en eminent test af, om specifikationen er komplet og korrekt. Hvis det giver problemer, vil der med garanti opstå fejl, når systemet bygges rigtigt.

Eksterne ressourcer til at bygge modellen sparer projektgruppens eget tids- og personforbrug. I modsætning til den normale fremgangsmåde hvor projektgruppen selv bygger en prototype, som derefter testes af andre. Det er ikke kun en kvantitativ forskel. Det bliver i det hele taget en anden test, og grebet rigtigt an, også én med kvalitative fordele.

7.24 Tænke-Højt-Test

Det er et middel til at *finde flere* fejl. Det kan gøres på to forskellige tidspunkter i et projektforløb. Det mest almindelige er, når det færdige system kan afprøves. Og på trods af alle de røster der måtte rejse sig om, at det er alt for sent, må vi dog lige pointere et par ting.

For det første at der også i den overskuelige fremtid, er behov for test. At ikke alt kan forhindres og klares præventivt. I så fald er THT god til at finde fejl i det færdige system. Der er desuden visse typer fejl i samspillet med forretningsgange, kundebetjening, ergonomi, etc. der er billigst at fange på det tidspunkt.

Det andet tidspunkt at køre THT på, er efter analysen. Det er så enten en variant af fortolkende reviews, eller den konkrete måde at teste en prototype på.

Pris?

Billigt. Der er ingen særlige forudsætninger, der skal være til stede. Der kan kun køres THT, hvis der er noget at køre den på. Det kræver et færdigt system eller en prototype, så det er dér, udgiften ligger. Selve THT'en koster kun tiden til selve seancen. Og den omkostning er meget ofte sparet ind, allerede når seancen er slut. I referenceprojektet vil det koste er par dages arbejde.

7.25 Structure Correlation

Et middel til at *undgå* fejl. Især den type der faktisk er helt overflødige, nemlig dem der opstår, fordi udviklerne ikke har et tilstrækkeligt kendskab til den virkelige verden. Altså har den nødvendige domæneviden. Kendere af den virkelige verden kan bibringe udviklerne den rette forståelse med det samme. Desuden er det nemmere for de samme kendere at finde fejl. Det er noget meget ligefremt, vi har med at gøre her. Hvis det system, der skal testes, er forståeligt for den, der skal udføre testen, er der større chance for at finde fejl. Hvis man kan genfinde alle funktioner og hændelser i et system, sådan som man kender dem fra dagligdagen, og med de samme navne, kan man bedre afgøre om de er rigtige.

Pris?

Når man bestemmer sig fra start, er der ingen omkostninger ved det. Hvis man skal designe et system om, er prisen afhængig af, hvor langt tilbage i et forløb det skal gøres.

7.26 Køb testen

Køb en ekstern test hos et konsulentfirma. De stiller testere til rådighed, der griber det professionelt an, og er trænede i metoder, teknikker og værktøjer.

Der er alle de muligheder for et 0-fejl forløb, som der er penge til. Mange er veget tilbage for tanken første gang fordi: "Hvad skulle de vide om vores system". Senere kan man ikke undvære det.

Pris?

Erfaringerne fra USA, hvor det gøres meget, siger at der skal betales for ca. 2 måneders arbejde på referenceprojektet. Det er den tid, der er nødvendig, for at eksterne testere kan følge med i hele udviklingsforløbet, og stå inde for at 0-fejl forløbet er troværdigt. Projektet skal stadig bruge de samme ressourcer på intern test.

For udgiften får man dog også det amerikanerne kalder 'skills transfer'. Det betyder at ens egne folk lærer principperne i et konkret forløb. Det er en god måde at købe undervisning på.

7.27 Brug af testorganisationen

Tiltaget forudsætter at der er oprettet en testorganisation. Der er mange løsninger i testorganisationens egen placering, lige fra selvstændig afdeling med reference til direktionen, og til sekundære placeringer i udviklings- og teknikafdelinger. Fordelen ved en høj placering er at budgettet afgøres af direktionen. Der er mulighed for at sætte strategier og mål i direkte sammenhæng med budgettet.

For et projekt handler det om at bruge testorganisationen, og om at bruge den mest muligt. På det operationelle plan har vi behov for et testmiljø med testtabeller og mulighed for at integrere løbende med andre systemer. Ligeledes på det operationelle plan kan vi have behov for at oprette testdata ved at hente fra produktionstabeller, og bruge dem efter den nødvendige anonymisering.

På det taktiske niveau vil vi også gerne bruge testorganisationen. Både i planlægningen og gennemførelsen af test. Alt hvad vi kan få af støtte er velkommen.

På det strategiske niveau har vi forbehold. Det kunne omfatte testorganisationens deltagelse i at afgøre om testen var god nok til at systemet kunne releases. Men deres råd og vejledning i beslutningen sætter vi pris på.

På det praktiske plan kan en testorganisation være af uvurderlig nytte. Den kan levere viden om årsager til fejl, viden om hvad andre projekter har gjort, kompetence om brugen af testværktøjer, etc.

Det er i høj grad muligt både at *undgå* fejl, og *finde flere* fejl.

Pris?

Jamen, det sparer os for tid og penge. Det er kun størrelsen af gevinsten der kan tales om. Og den styrer vi selv, ved at bestemme hvor tidligt i projektet vi starter samarbejdet. Jo, tidligere jo bedre, og jo større bliver besparelsen.

7.28 Fejlårsagsanalyse

Sådan lidt frit oversat fra Defect Causal Analysis. Kausalanalyse drejer sig om at forstå sammenhænge og at finde den egentlige årsag til en konstateret fejl.

Analysen kan både bruges til at *undgå* fejl, og *finde flere*.

Når et projekt er slut, er der stor værdi for organisationen i at identificere de væsentligste årsager til de typer af fejl som vi gerne havde været foruden. Dem der har været besværlige at rette, samtidig med at der har været en vis mængde af dem.

Der er også stor værdi for projektet, ved at analysere årsager under vejs. Hvis vi kan konstatere at en væsentlig del af de besværlige fejl vi finder efterhånden som integrationen skrider frem, stammer fra at udviklerne misforstår indholdet af Use Cases, ja så er der grund til at tage sig af det. Enten ved at reviewe og forbedre de Use Cases der udestår, eller at checke hvordan det er gået med dem der er udviklet men endnu ikke integreret. Afhængig af hvor langt projektet er nået, kan der sættes ind med forbedringer.

I 'gamle dage' testede man troligt derudad uden at analysere på fejl. Mere eller mindre ubevidst på en formodning om at hver fejl var en selvstændig hændelse. Men fejl har årsager, og jo tidligere de kan findes, jo tidligere kan årsagerne fjernes.

Pris?
Der er mange praktiske erfaringer på dette område, og hvis man kan ramme et fornuftigt niveau for analysen, kan man nøjes med at bruge en halv time per fejl. Nøjes med at vælge dem ud der er værd at gå efter. Lad være med at analysere på alle, tværtimod, vælg dem der giver en positiv cost/benefit hvis vi kan fjerne eller reducere forekomsten af dem. Lad også være med at analyse for langt tilbage i hvad der er den egentlige årsag. Enhver årsag kan føres tilbage til Adam og Eva, og at han skulle have ladet æblet være. Men der er for eksempel mere brug for at konstatere at årsagen til nogle fejl er svært forståelige Use Cases.

7.29 Integreret sporbarhed

Med almindelig sporbarhed menes at ethvert krav kan føres tilbage til interessenten der har stillet det. Med integreret sporbarhed føres princippet videre i testen. Det er en del af moderne udviklingsmetoder, for eksempel Rational Unified Process, at vi helt ned til og med den enkelte testcase kan spore den til en Use Case, der igen kan spores tilbage til et krav der igen kan spores tilbage til en interessent. I al sin enkelhed består det jo blot i at knytte de forskellige dele til hinanden. Men arbejdet med at sammenknytte skal gøres, før vi kan nyde godt af det. Derefter giver det os store fordele når vi skal teste sammenhængende krav, og når vi skal involvere interessenterne i præcis den del af testen som de skal godkende. Det er også et middel til at give os sikkerhed for at alle krav er dækket af test.

Tiltaget er med til at *finde flere* fejl, fordi det giver en bedre dækning.

Pris?
Det er gratis. Det koster ekstra tid at være opmærksom på hierarkiet, og det koster tid at knytte forbindelserne. Men i det store og hele opvejes det af de timer vi sparer til i modsat fald at checke om vores testcases er dækkende.

7.30 'Defensive Programming'

Det er et middel til at *finde flere* fejl, og til at være opmærksom på dem noget før. I og for sig er det en afledt fordel, af det egentlige formål, som er at overleve når der opstår fejl i brugen af systemet.

Defensive Programming går ud på at koden forsvarer sig. Den checker selv om det ikke er nødvendigt. Alle muligheder undersøges og håndteres, også selvom de angiveligt ikke kan opstå. Alle data checkes som om de kunne indeholde invalide værdier, også selvom det angiveligt ikke er muligt. Defensive Programming har været brugt i mange år, men stort udelukkende inden for Mission Critical software. Hvis koden skal styre åbningen og lukningen af ventiler på et kraftværk, så er det meget godt at det ikke sætter sig på halen midt i det hele.

Når koden indrettes med Defensive Programming, giver det os en fordel i testen. Fejlene kan på det nærmeste beskrives af koden, i tilfælde hvor vi måske først ser resultatet af fejlen et andet sted i systemet. Men det giver også en forskel for programmørerne. De bliver mere opmærksomme på fejlmulighederne, og kan finde fejl i det grundlag de arbejder på, når de er mere åbne over for at det kan være tilfældet.

Pris?

Vi har forskellige erfaringer med prisen. Tag udgangspunkt i 10 % mere tid til kodning, og forvent mere eller mindre, afhængigt af udviklernes ambitionsniveau.

7.31 Test som mikrodesign

Her er en metode til at designe de enkelte dele af systemet. Metoden udspringer af Extreme Programming, hvor planlægning og udformning af testdata bruges til at drive et udseende frem og drive en konkretiseringsproces. Test er ikke passiv afprøvning men en aktiv del af udviklingen. De enkelte dele af testen konkretiseres i testcases, og man er nødt til at fastlægge systemets detaljerede design for at kunne skrive fysiske testdata.

Forudsætningen for dette tiltag er at der skrives testdata før kode. Man kan derfor vælge at se tiltaget som en del, eller en forlængelse, af tiltaget af samme navn. Men mikrodesignet rækker længere og har nogle flere forudsætninger. Det ændrer udviklingsprocessen og blandt andet bruges der mere tid på test. Der er brug for mere tid til at diskutere design samtidig med testplanlægning. En væsentlig forudsætning er at kunden skal være til stede i projektet, det der i XP er dækket af princippet om 'On-Site Customer'.

Tiltaget er med til at *undgå* fejl, og det er et væsentligt middel til at få fejlfrie programmer i første hug.

Pris?

Det er svært at bedømme fordi arbejdsprocessen i projektet ændrer sig så meget, og der ikke er gennemført kontrollerede sammenlignende forsøg. Alene bedømt ud fra erfaringer er det her et gratis tiltag.

7.32 'Pair Testing'

Se beskrivelse af 'Pair Testing' i et andet appendiks om 'Extreme Programming'. Tiltaget skal med her fordi det er et af de nemmeste at sætte i værk, og et tiltag der er meget effektivt på området *finde flere* fejl. Det består i al sin enkelhed i at testerne arbejder i par, som det kendes fra 'Pair Programming'. Det øger motivationen og gejsten, og testen bliver markant bedre.

Pris?

Der mangler kontrollerede forsøg, men tag udgangspunkt i tallet fra 'Pair Programming' på at der bruges 15 % flere timer. Det er det der skal tjenes et andet sted i processen for at få en nytteværdi.

7.33 Procesforbedring (TMM)

Vi taler her om Testing Maturity Model. Se modellen i detaljer på http://www.softwaretest.dk/. Det er en forudsætning at modellen er implementeret i organisationen, så den er brugbar for projektet. Modellen bruges til at dele testprocessen op i områder der kan forbedres enkeltvis, og niveauerne i modellen giver en fremgangsmåde.

Pris?

Der er ingen omkostninger i det enkelte projekt. Omkostningerne ligger på virksomhedsniveau, når modellens indhold implementeres.

7.34 Paralleltest

Behøver man egentlig at sige mere? Som et middel til at sikre 0-fejl, kan man checke facitlisten.

Man kan kun undre sig over, at det ikke gøres mere end tilfældet er. At alle systemer ikke går i luften med 0-fejl, efter en paralleltest der sikrer det rigtige resultat. Men det er der nogle grunde til:

- Det er dyrt at arrangere paralleltest. Indsatsen kan være dyrere end det udbytte man tror på.

- Det kan være svært at finde nogen, der kan afgøre, hvordan det gamle system virker. Verden er splittet op i specialister uden det forkromede overblik.

- Det kan være svært at afgøre de eksakte sammenhænge mellem det gamle og det nye system. Specielt hvis det gamle er manuelt. Men selv når det er et IT-system, der afløses, kan der mangle entydige én til én forhold.

Man må selv afgøre om det er undskyldninger eller gode grunde.

Pris?

Prisen bliver ofte et problem, når der skal køres parallelt i en så lang periode, at det er troværdigt. Omkostningerne er store, fordi paralleltesten skal ligne rigtig drift. Det kan handle om at have to store systemer kørende samtidig.

Men lad os slå fast at udsagn som: ”Det er simpelthen ikke teknisk muligt at køre parallelt” er usande. Det kan blot være økonomisk utiltalende. Problemet er dog sommetider, at de beregninger der viser økonomien, ofte bygger på antagelser om, at det nye system er mere korrekt, end det viser sig at være.

7.35 Beta-test

Lad dette tiltag blive det sidste fordi det er det sidste middel til at sikre et fejlfrit produkt. Det er test i produktion, således at en eller flere brugere påtager sig at anvende systemet rigtigt. Hvis der er fejl, gør det ondt, fordi instrumenterne er skarpe.

Pris?

Prisen ligner den foregående på paralleltest. Det kan arrangeres som en almindelig ibrugtagning blot med et begrænset antal brugere. Det giver en test af igangsætning som sidegevinst.

Beta-testen gennemføres ikke altid fuldt ud, selv i tilfælde hvor nytten er åbenbar, på grund af pressede tidsplaner. Når projektet nærmer sig slutdatoen, er det for sent at tale om at køre en beta-test på et par måneder. Den skal være planlagt fra første færd og respekteres, så den udføres i sin planlagte udstrækning.

Mangler der noget?

Måske er der nogen der savner lige præcis deres yndlingsteknik. Det kan der være grunde til. Mange teknikker er udeladt, fordi de i 0-fejl sammenhæng kun er midler til at få sat tiltagene i værk. Det betyder at mange testteknikker, inklusive reviews og inspektioner, ikke optræder på listen.

8. Ledelsens opgaver

0-fejl er først og sidst teknik. Det er ikke nok at bede ledelsen om opbakning, og om at komme med signaler etc. Man skal præsentere en testplan der foreslår nogle tiltag og fortælle, hvad det koster. Suppler den med alternative ambitionsniveauer og budgetter. Kort sagt, sørg for en dialog. Det er testerens opgave at være klædt på til at stå inde for de faglige og tekniske muligheder. Både med hvad der kan lade sig gøre, og hvad det koster.

Ledelsen skal stilles overfor nogle attraktive muligheder, med klar gennemskuelig økonomi. Derefter er det ledelsens opgave at:

- stille krav om pålidelighed
- stille penge og ressourcer til rådighed.
- måle brugernes tilfredshed.
- beregne de totale omkostninger ved fejl.
- få testmodellen gjort tilgængelig og interaktiv via et intranet
- bede om fejlanalyser på alle afsluttede projekter
- bede om regelmæssige fejlanalyser på systemer i forvaltning
- bede om status på testmiljøet
- insistere på at vedtagne procedurer bliver fulgt

Ledelsens indsats skal ikke være en politi- og overvågningsfunktion. Men selvfølgelig skal den stille krav og følge op.

Ledelsen skal f.eks. som nævnt stille krav om, at vedtagne procedurer bliver fulgt. At udviklings- og testmodeller bruges. Således at vedtagne retningslinier for test får den virkning, de er tiltænkt. Den simpleste måde er at lægge det ud på et intranet. Opgaveansvarlige kan slå op i systemet, for at se og følge kravene. Det er også nemmere at ajourføre kravene centralt.

Ledelsen har også den regnskabsmæssige opgave. Når regnestykket skal gøres op, og det viser sig om der har været en nytte af investeringerne i bedre test. Det kan gøres ved at benchmarke i forhold til tidligere år, og til det formål skal der bruges målinger og metrikker.

9. Valg af tiltag

Er nogle af tiltagene vigtigere end andre, og måske ligefrem kritiske succesfaktorer. Svaret er nej. Det situationsbestemte skal vinde over teori og blind tro.

Test handler om facts, ikke om religion. Der er brug for folk, der reelt kan finde fejl, og reelt kan sige god for et produkt. Der er ikke brug for ideer om, at bestemte metoder er bedre end andre. Og slet ikke at en enkelt ny metode er så god, at den af sig selv giver rigtige produkter. Det er muligt den kommer en dag. Men så skal den udnævnes til det af en tester, der har konstateret det. Ikke af en udvikler der ønsker det.

En anbefaling ved valg af tiltag til 0-fejl:

- Vælg mere end ét tiltag
- Vælg nogle der *undgår* at fejl opstår
- Vælg nogle der *finder flere* fejl

- Vælg tiltag der er 'sjove', det vil sige motiverende for testerne
- Vælg noget der involverer forskellige personer
- Vælg tiltag der tester produkter
- Vælg tiltag der tester processen
- Vælg noget der har effekt på kort sigt
- Vælg noget der har effekt på lang sigt

Projekter og forvaltningsopgaver
For projekters vedkommende er det opgavens størrelse og sværhed, der spiller ind. Ét projekt varer måske et måned, et andet varer år. Hertil skal lægges variationen i kompleksitet.

Forvaltningsopgaver skal vurderes endnu mere nuanceret. Ud over opgaven skal der tages hensyn til systemet der gribes ind i. Selvom en opgave er lille, kan testen blive omfattende. Hvis kravet er 0-fejl, betyder det at hele systemet skal testes for evt. afledte fejl. Behovet er dermed ikke kun bundet til opgavens størrelse, men også til systemets kvalitet.

Desuden skal der tages hensyn til typen af forvaltning. Forudsætningerne for 0-fejl er forskellige i en saneringsopgave, i forhold til at rette en fejl, tilpasse til en ændret lovgivning eller udbygge med nye funktioner.

Set fra et traditionelt synspunkt er der til forvaltningsopgaver især brug for 'Automatisk regressionstest' og 'Paralleltest'. De er nyttige for alle fire forvaltningskategorier. Men det er en passiv tankegang at stoppe der. Suppler med 'Verifikation og Validering parallelt', 'Involvering af interessenter', 'Refactoring frem for reparation' og 'Prototyper'. Men der vil altid være behov for at vælge våben i hvert projekt og i hver forvaltningsopgave. For at vinde krigen over fejl.

10. Effekt og pris

Definitionen på øget produktivitet i et testforløb er slet og ret at *undgå* og *finde flere* fejl. Figur 1 kan bruges til at vælge. Figuren viser tiltagene med markering af effekt og pris. Det er klart, at konkrete forhold i et projekt kan ændre billedet. Men hvert tiltag er vurderet således:

U = Medvirker til at *undgå* fejl
F = Medvirker til at *finde flere* fejl
G = Gratis
T = Omkostningen er tid
I = Investering er nødvendig

Tiltag	U	F	G	T	I
1. Offensiv strategi	√			√	
2. Verifikation og validering parallelt	√			√	
3. Testenheden fastlægges før/under analysefasen	√		√		
4. Design med test i tankerne	√			√	
5. Alle farver test		√	√		
6. Testleder i projektet	√	√	√		

7. Testere i projektet	√	√		√	
8. Testdesign før systemdesign	√		√		
9. Testdata før kodning	√			√	
10. Test af test		√		√	
11. Seriøs udviklingsdokumentation	√			√	√
12. Testspecifikation udvikles seriøst	√	√		√	
13. Box Structured Design	√			√	
14. Automatisk regressionstest		√			√
15. Holdninger	√	√	√		
16. Standarder	√	√	√		
17. Dækningskontrol		√			√
18. Refactoring frem for reparation	√			√	
19. Testplanlægningsseminar	√	√		√	
20. Erfaringsopsamling (Knowledge Management)	√	√		√	
21. Involvere interessenter	√	√	√		
22. Halsted & McCabe	√	√			√
23. Prototyper	√	√			√
24. Tænke-Højt-Test		√		√	
25. Structure Correlation	√			√	
26. Køb testen	√	√			√
27. Testorganisation	√	√			
28. Fejlårsagsanalyse (Defect Causal Analysis)	√	√		√	
29. Integreret sporbarhed		√			
30. Defensive Programming		√		√	
31. Test som mikrodesign	√		√		
32. Pair Testing		√		√	
33. Procesforbedring (TMM)	√		√		
34. Paralleltest		√			√
35. Betatest		√			√

Figur 1: Tiltag

11. Test af ændringsønsker

Ændringsønsker skal gennemføres fra det rigtige punkt i forløbet. Det vil sige at en ændring, der modtages i konstruktionsfasen, men som har effekt for designet, betyder at man må gå tilbage og omgøre den nødvendige del af designarbejdet. Ikke af hensyn til den smukke teori, men af simple driftsøkonomiske grunde. Til at styre ændringsønsker aftales på opstartseminaret hvilken konkret procedure, der skal følges. En såkaldt 'Change Control Procedure' eller CCP. Den er vist i figur 2 i en simpel udgave.

I gamle dage blev en projektleder belønnet for at kunne håndtere ændringsønsker dynamisk og uden at formalisere det for meget. Det betød ustabile løsninger. I vore dage er der større lyst til, at vide hvad der foregår, og at aflevere til tiden. Derfor skal CCP'en bruges, og som en konsekvens skal det testes, at den bliver det. Ellers er der åbnet for et smuthul i et ellers velplanlagt 0-fejl forløb.

1. Dokumentér det specifikke ønske
2. Registrér det på en log eller i en database
3. Afgør hvor langt tilbage der er effekt
4. Beregn Cost/Benefit
5. Beregn konsekvenserne for planen
6. Indstil til styregruppen
7. Informér interessenterne

Figur 2: Behandling af ændringsønsker

Test at CCP'en bruges
En almindelig passiv test kan iværksættes ved ganske simpelt at undersøge, om der har været ændringer, og hvordan de er blevet behandlet.

Det kan gøres ved hver milepæl eller faseskift. En mere aktiv og præventiv test kan iværksættes i starten af en fase, ved at teste om både udviklere og interessenter ved hvordan kommende ændringsønsker skal behandles. Om de ved at CCP'en findes, og hvad de skal gøre, når de har brug for den. Testere skal være opmærksomme på to forhold i forbindelse med ændringsønsker:

- Test af at CCP'en bruges korrekt
- Test af den gennemførte ændring

De to tests bliver ofte udført af forskellige personer. Testeren kan selv teste at CCP'en bruges korrekt (verificering). Det kan derimod være en bruger der tester (validerer) at ændringsønsket implementeres fagligt korrekt.

12. Gør op med myterne

I et 0-fejl forløb må man gøre op med nogle myter. Den første øvelse bliver at tilslutte sig de holdninger, der ligger til grund for appendikset 'Om fejl'. Derefter at vælge de tiltag i figur 1 der giver størst nytte. Tilsammen fører de to øvelser til et opgør med myterne.

Myte 1:
"Det er ikke muligt at fremstille en kravspecifikation, der er komplet, korrekt og stabil. Verden ændrer sig, og kravene til systemet skal ændres løbende. Vi får heller ikke lov at indarbejde ændringer ordentligt. Det er der simpelt hen ikke tid til".

Opgør med myten
Selvfølgelig ændrer verden sig. Men så skal kravspecifikationen også gøre det. Gå tilbage til det sted i forløbet hvor ændringen optræder første gang, og indarbejd den som om den

hele tiden havde været der. Glem alle de indsigelser, der ligger på tungen, om at det ikke kan lade sig gøre. Bare gør det. Ret funktions- og databeskrivelser og bede om en validering af, at det er den ønskede ændring.

Hvis det direkte forbydes af projektlederen, er der selvfølgelig ikke noget reelt ønske om at undgå fejl. Så er der ikke noget at gøre ved det. Men ellers er det projektlederens opgave at bruge en CCP, der i simple trin foreskriver, hvordan ændringsønsker skal behandles i det pågældende projekt/firma. Tag nu ordentlig fat: Det er helt åbenbart at en mangelfuld indarbejdning af ændringer i en kravspecifikation, stammer lige så meget fra udvikleres manglende lyst til at gøre det, som fra ledelsens, herunder projektlederens forbud mod at gøre det.

Det er uprofessionelt at lade være med at indarbejde ændringer ordentligt. Specifikationen *er* systemet. Koden er der kun, fordi maskiner er for dumme til at læse specifikationer, og må arbejde med en oversættelse.

Myte 2:
"En godkendt (analyse/model/rapport) er en fejlfri (analyse/model/rapport).

Opgør med myten
Ingen tror på myten, alligevel eksisterer den. I hvert fald lægges der stor vægt på at arbejde videre med en godkendt specifikation, godkendt model, godkendt prototype, etc. Men enhver ved samtidig, at de har set læssevis af godkendte specifikationer og modeller, der på ingen måde var fejlfri. Fejlfrihed kommer igennem hårdt arbejde, ikke ved at dække sig ind under at noget bør være fejlfrit, fordi det er godkendt. Testere har *ansvar* for at noget er fejlfrit.

Myte 3:
"I et iterativt forløb kan man ikke forvente fejlfrie specifikationer"

Opgør med myten
Iterationer er netop et middel til fejlfrihed. Hvis man ikke er sikker på indhold og mål, foretages iterationer mellem faserne. Netop for kun at fastlægge de dele der kan stoles på. I de første iterationer fastlægges tingene på et overordnet niveau, fordi detaljerne kan ændre sig. Men det der fastlægges, skal være komplet, korrekt og stabilt. På det niveau. I næste gennemløb fastlægges yderligere dele, der kan stoles på.

Iterationer er ikke en lejlighed til at være sløset eller uopmærksom. Eller en mulighed for at lade fejl blive i en specifikation, fordi den alligevel ændrer sig.

Myte 4:
"Fejlfrihed er for dyrt, og den sidste fejl er for kostbar at finde"

Opgør med myten
Mellemresultaterne i et forløb skal være komplette, korrekte og stabile. Det er målet, og jo mere det holdes for øje, jo bedre bliver slutresultatet. Det bliver først rigtig dyrt, hvis målet er et andet.

Derfor skal der vælges tiltag på et andet grundlag end myternes. De praktiske begrænsninger og bestemmende forhold når man studerer listen over tiltagene i figur 1, skal lyde således:

- Nogle af dem bruger vi allerede
- Nogle af dem kan vi med fordel umiddelbart begynde at bruge
- Nogle af dem kan vi med fordel lære os at bruge
- Nogle af dem er ikke relevante for os
- Nogle af dem er for dyre, selvom de er relevante
- Nogle af dem har så mange forudsætninger at vi snubler i starten

Gør op med myterne og vurder i stedet de praktiske muligheder.

13. Test og kvalitetsstyring

I et moderne testforløb er det svært at se hvornår det ene holder op og det andet begynder. Sådan bør det i hvert fald være. Test- og kvalitetsstyring skal arbejde sammen, forbedre hinanden og have en synergieffekt på forløbet.

En projektleders syn på en opgave skal være holistisk, det vil sige baseret på et helhedssyn. Det duer ikke at dele verden op i discipliner, svarende til de kurser man har deltaget på, eller de bøger man har læst. Det duer heller ikke på virksomhedsniveau. Kvalitetsstyringsfunktionen skal påvirke testforløbene. I praksis. Og hver fejl, der rapporteres, skal bruges den modsatte vej til at forbedre kvalitetsstyringen, fordi den på det punkt har svigtet.

Det er heller ikke nok at holde seminarer om kvalitet. De er gode til at påvirke holdninger, og til at give signaler og budskaber. Men de skal følges op af handling.

Det holistiske syn betyder at testforløb kan nyde godt af de erfaringer kvalitetsfolkene stiller til rådighed. Det betyder også at både test og kvalitetsstyring bliver en investering. Og ikke en omkostning.

Helhedssynet giver mulighed for nye tanker. Og de er en forudsætning for at handle anderledes.

Kvikstart

Stil følgende spørgsmål: Har vi styr på kvaliteten af vores produkter. Forstået på den måde at den kan forudsiges. Ellers vedtag et testforløb med indlagte 0-fejl tiltag. Det er ikke sikkert at det kan betale sig at gå efter 0-fejl. Men den ambition, der formuleres, skal der være sikkerhed for at nå.

Implementér forløbet maskinelt og interaktivt. Det skal gøres under ansvar af en evt. kvalitetsstyringsfunktion, eller af en kvalitetsgruppe med ansvar og kompetence.

Formulér et program for kontinuerlig forbedring, der bygger på erfaringerne fra afsluttede testforløb.

Tilbage til toppen af tekst