Om fejl

Appendiks til bogen ’Struktureret Test’

1. Hvad er den rigtige fejl?

Fejl, defekter eller bugs. Det er det samme vi taler om. Det kan være en mangel, noget overflødigt, noget uklart eller noget forkert. I det følgende bruges 'fejl' om det hele.

Lad mig bruge et ultrakort eksempel. En brugertester er nået til den gennemførende testfase på det samlede kørende system. En testcase med et forventet resultat på kr. 100,- viser et reelt resultat på kr. 101,-. Efter at have overbevist sig om at testmaterialet er korrekt, udfylder testeren en fejlrapport. Den beskriver en fejl i uddatamaterialet.

Projektlederen er imidlertid ikke tilfreds. Det er ikke den rigtige fejl, der er blevet beskrevet. En programmør bliver bedt om at redegøre for årsagen. Den findes; det viser sig at være rækkefølgen af faktorerne i en beregning. I to af modulerne er den ikke korrekt. Der udfyldes en ny fejlrapport, der beskriver fejlen, og hvad der skal rettes. Vi har nu to meget forskellige beskrivelser af fejlen. For at gøre det endnu besværligere antager vi, at der kobles en konsulent på projektet, der uden tøven forkaster de fejlrapporter, der er udfyldt. Årsagen er efter konsulentens mening en brist i kommunikationen mellem brugere og udviklere, og fejlen er at systemspecifikationen ikke fastslår rækkefølgen af faktorerne.

Hvad er den rigtige fejl: Den forretningsmæssige, rækkefølgen af nogle instrukser, en mangel i specifikationen, eller alle tre?

2. Hvornår er en fejl alvorlig?

På fejlrapporter skal der sættes et kryds for om en fejl er alvorlig, almindelig, eller en uhensigtsmæssighed. Hvad er en alvorlig fejl?

Én mulighed: når den vil koste forretningen dyrt, hvis den slipper igennem til daglig drift. En anden mulighed: når den er besværlig at rette.

Hvis begge betingelser er opfyldt, må den siges at være alvorlig. Men hvad nu hvis den kun opfylder én af dem?

Hvis vi vælger den første definition er de 101,- kroner en alvorlig fejl hvis det sætter kunden i forlegenhed, skaffer os på midtersiderne af aviserne, eller ikke opfylder nogle bestemmelser i loven. Men den kan godt være nem at rette.

Hvis vi vælger den anden definition er det en alvorlig fejl, hvis det kræver omfattende reparationer i syvogtyve moduler. Selvom det er en statistisk oplysning, hvor det ikke betyder så meget om det er 100,- kr. eller 101,- kr.

3. Taksten på reparation af færdige systemer

Når der skal sættes en pris på at rette de 101,- kr., slår vi op i taksttabellen for reparationer. Nævnt fra billigste til dyreste:

- Der er fejl i en eller flere instrukser, men det er kun i koden, ikke i grundlaget.
- Der er fejl i grundlaget for koden, men det er kun funktioner der skal rettes. Der er ingen ændring i datas behandling og transformering. Rettelsen giver ikke ændring i de opbevarede data.
- Der er fejl i den måde attributterne håndteres og transformeres på. Dataværdierne er forkerte. Men selve entiteterne/klasserne er gode nok.

- (det var den billige afdeling, nu bliver det dyrt)

- Der er fejl i entiteter/klasser. Reparationen omfatter ændringer i en eller flere af dem. Men deres relationer er gode nok.

- Der er fejl i relationerne mellem entiteter/klasser.

- Der er fejl i designet af entiteter/klasser.

4. Reparationens omfang

Fejlen skal udbedres. Først er symptomet opdaget, siden er fejlen fundet, nu skal systemet repareres. Det giver muligheder for nye fejl. Den første mulighed er afledte fejl, når der ændres i systemet. Resultatet er, at der opstår en ny fejl, der skal beskrives. ”Hvor fanden kommer den fra”, tænker vi, når den viser sig. Den anden mulighed er, at reparationen ikke er tænkt 100 % igennem. Resultatet er, at fejlen viser sig igen, bare lidt anderledes. ”Hovsa, det var ikke lige sådan”. Den tredje mulighed er, at der kun repareres delvist, og fejlen får lov at blive. F.eks. når den korrekte rækkefølge af faktorerne ikke tilføjes specifikationen. Så er der frit slag for andre, der bruger specifikationen til at begå flere af samme slags. ”Den troede jeg vi havde rettet”, siger vi, når vi ser den. Den fjerde mulighed er, at oplysningen er givet i troen på, at den var rigtig. Så skal opfattelsen hos den, der skrev specifikationen, ændres, ellers vil fejlen blive gentaget i nye specifikationer. ”Åh nej”, tænker vi, hvornår lærer vi af vores fejl.

Reparationens trin

Reparationer skal udføres med omhu i begge dens to hovedfaser:

- analyse af fejlen
- udførelse af reparationen

Split det evt. yderligere op i de nødvendige mindre trin for at være sikker på et vellykket resultat. Brug de 14 trin i en forvaltningsopgave som grundlag, men skræddersyet til en reparation:

1. Analyse af hvad der er den egentlige fejl
2. Analyse af den nødvendige reparation i systemet
3. Design af reparationen i form af de enkelte steder der skal rettes
4. Analyse af behov for afledte rettelser i andre systemer
5. Markér rettelse på kildetekst
6. Afhold et kodereview på rettelsen
7. Indfør rettelsen i kildeteksten
8. Modultest af rettede moduler (White-Box)
9. Integrationstest af rettede grænseflader (White-Box)
10. Systemtest i demomiljøet
11. Brugertest i demomiljøet
12. Dokuménter reparationen, helst i en database
13. Igangsætningsreview på det samlede forløb omkring fejlen
14. Sæt det rettede i drift

5. Test kontra reparation

Test foretages af testere med henblik på at finde fejl. Reparation foretages af udviklere for at fjerne en fejl. Reparation og debugging er det samme, og har samme formål: at fjerne fejlen. Hvis den tidlige og løbende test forsømmes, fører det til mere debugging af det færdige system. Det kan endda føre til, at programmørerne bliver testere. Det sker, når de bliver bedt om at finde og forklare alle fejl, uanset hvilken fase de er begået i. Lad os forestille os et sådant ineffektivt forløb. Den forkerte specifikation af rækkefølgen viser sig at være én blandt flere mangler og uklarheder. Brugertesteren finder nye fejl, og opretter nye fejlrapporter. Årsagen til hver fejl bliver troligt fundet og rettet af programmørerne. Sådan fortsætter det med BRUGERTEST - DEBUGGING - PROBLEMLØSNING. Men nogle af fejlene stammer fra faser der ligger før konstruktionen.

En behjertet sjæl siger: Egentlig burde vi revidere specifikationen. Ordet 'egentlig' bruges allerede af den behjertede, i erkendelse af svaret: "Det har vi altså ikke tid til lige nu. Rettelse af dokumentationen må vente til senere".

Men pointen er forkert. Miseren er at de egentlige fejl ikke rettes. Det er langtfra kun et dokumentationsspørgsmål. Der sker værre ting. De fleste reparationer går godt, men en vis mængde går skidt. Der udtænkes nogle forkerte og ukomplette løsninger. Der opstår afledte fejl, fordi en analysefejl kun kan udbedres ved en fornyet stump analyse. Følgefejl kan derfor let overses. Der bliver brug for mere test, debugging og reparation. Og nogle af fejlene slipper helt igennem til drift.

Sådan kan forløbet blive, hvis fejl og mangler i specifikationen ikke sendes tilbage til specifikationsfasen og repareres der. Det bør de blive, Ikke kun fordi det er teoretisk korrekt, men fordi det *er* det korrekte. Men helst skulle de have været fundet og repareret, før fasen blev endelig godkendt.

Det ineffektive forløb har også andre ulemper. Det betyder at programmørerne bruger tid på fejl, som de ikke har begået. Selvfølgelig skal de analysere deres egne fejl og bruge tid på det. Og de skal også, om nødvendigt, hjælpe med til at afgøre, hvad der er de egentlige fejl. Men de skal ikke forklare og løse alle fejl. Debuggingen af det færdige system bliver for omfattende, og arbejdsbyrden på programmørerne stiger. Tidsplanen kan let skride, og totalt set bruges der mere tid på test, end der havde været nødvendigt.

Mængden af reparationer i de tidlige faser er en indikation af hvor meget test der er nødvendig for den samme del af systemet i de efterfølgende faser. Hvis en del af analysen kan erklæres fejlfri, med få reparationer, efter en nøje gennemgang og afprøvning, kan der skrues ned for blusset i de følgende testfaser. Hvis der derimod har været mange ændringer, reparationer og fejlrettelser i et afsnit, skal der sættes ind med systematisk afprøvning af det samme område i de efterfølgende faser.

6. Hvornår opdages fejl?

Første mulighed er at opdage fejlen, før den kommer på tryk. Det kan opnås ved at tænke før der tales, tale før der skrives ned, og overveje tingene med andre før der træffes beslutninger. Masser af fejl begås men opdages, og det ville da være dejligt hvis alle fejl blev fundet på den måde. Den type fejl registreres selvsagt aldrig på en fejlrapport.

Anden mulighed er at finde fejlen med det samme den er kommet på tryk. Enten ved at kontrollere det selv en ekstra gang, eller bede andre om at gøre det. Det

kan vise sig at en telefonopringning er en god investering, før et produkt går videre i processen. Fejlen lever i meget kort tid. Men engang imellem er man nødt til at tegne en streg på papiret for at se at den er gal. Den skal bare viskes ud med det samme. Fejlen bliver ikke registreret på en fejlrapport.

Tredje mulighed er at finde fejlen i den fase den begås. Reviews og inspektioner finder fejlene, før fasens produkt godkendes. Definitionen på 0-fejl udvikling ligger ofte på dette niveau: Når hver fase afsluttes med fejlfrie produkter. Der kan tales for og imod at registrere disse fejl formelt. Hvis man er interesseret i fejlanalyse skal der tales for.

Fjerde mulighed skal der helst ikke være nogle af. Det er V-mekanismen der træder i kraft. Hvis fejlen slipper igennem en udviklingsfase, findes den først i den modstående testfase. Man kan håbe at de findes tidligere, men det gør de som oftest ikke. En fejl der slipper igennem analysen, bliver som regel først fundet i system- eller brugertest. Den bliver ikke fundet i designet, konstruktionen, modul- eller integrationstest. Det er overordentligt uheldigt. Den mentale kontrol er væk, folkene er måske væk. Det er lang tid siden, at projektet beskæftigede sig med det produkt, fejlen er begået i. Det er ikke muligt at vende tilbage på en studs. Vi har bevæget projektet et andet sted hen. Der bliver tale om reparationer i gængs forstand. Disse fejl skal omhyggeligt registreres.

Femte mulighed må ikke gerne forekomme. Det er når fejlen bliver fundet efter afslutningen af testforløbet. På et udefineret tidspunkt efter at systemet sat i drift. Registrér, analysér og lær! Der er grund til omhyggelig analyse af hver eneste af dem.

7. Omkostninger ved fejl

Omkostningerne ved en fejl er opgjort i figur 1. Det giver en forskel om fejlen sker efter at systemet er gået i drift. Så er alle omkostningerne i figur 1 mulige. Hvis det er en fejl, der kun eksisterer under udvikling eller test, falder nogle af dem væk.

Det er svært at reparere fejl. Det kræver håndværksmæssig kunnen, og det kræver omtanke at gøre elegant. Selv en reparation på en analyserapport, kan give problemer. Det kan ikke bare gøres ved at tilføje nogle linier i margenen. En fejl skal rettes, ved at reparere alle de steder i rapporten den har betydning. Hvis analysen ændres på nogle punkter, skal rapporten skrives om, så den ser ud, som den ville have gjort, hvis den havde været rigtig fra start.

Tiden der bruges på at begå den
Rapportering om at fejlen er begået
Tiden der bruges på at finde den
Tiden der bruges på at rette den
Tiden der bruges på at teste rettelsen
Konsekvensen af fejlen på data
Reparation af ødelagte data
Dokumentation af hvad der er repareret
Håndtering af systemet indtil det er rettet
Igangsættelse af ny programversion
Rapportering om at rettelsen er foretaget

Figur 1: Omkostningerne ved at begå en fejl

8. Hvor kan det betale sig at se efter fejl

Hans Schaefer har gode råd om hvor det kan betale sig at kigge efter fejl.
Se først efter specifikationsfejl:

> Èn kilde til fejl er, at brugere har ombestemt sig under vejs. Resultatet er ændrede specifikationer. Identificer de krav, der har været flest ændringer til, og test dem først.
>
> Har der været uenighed om kravene? Har der været forskellige ønsker blandt brugere? Er det et kompromis der ligger til grund for det videre arbejde? Kompromiser er ofte fejlbehæftede.
>
> Har der været organisationsændringer i brugerorganisationen? Undersøg de steder i kravene der er defineret af skiftende organisationer.
>
> Har der været organisatoriske stridigheder - to eller flere afdelinger ønsker forskellige faciliteter? Igen et kompromis. Find ud af hvilke afdelinger der er uenige, find ud af hvilke krav der er stillet af begge, og test dem.
>
> Undersøg kvalifikationerne hos de ansvarlige for specifikationen. Er det personer med tilstrækkelig viden? Eller er det nyansatte, der er sat på jobbet, fordi afdelingen ikke ønskede at bruge nogle af sine nøglepersoner?
>
> Har der været tid nok? Identificer de krav, der har været mest presset da de skulle have deres endelige udformning.
>
> Er alle spørgsmål, svar og forslag behandlet. Identificer de krav der ikke er behandlet hundrede procent færdigt.

Det næste område er designfejl:

> Identificer de grænseflader der oftest er blevet ændret.
>
> Identificer de designbeslutninger der har været mest ustabile.
>
> Er der dele af designet, der er udført af grupper med en høj personaleomsætning?
>
> Er der personer i udviklingsgruppen, der har haft personlige uoverensstemmelser? Det vil give ringere og mindre kommunikation.

I konstruktionsfasen:

> Identificer de største moduler, og moduler med høj kompleksitet.

Er der moduler, der først er defineret i konstruktionsfasen (Post Design)

Identificer de dele af systemet der måtte være skrevet af uerfarne.

Har der været personlige uoverensstemmelser som nævnt under designfejl?

Er der dele af koden, der ikke er reviewet?

Hans Schaefer bruger disse teknikker til at identificere de mest sandsynlige områder for fejls opståen. Ved at identificere områderne tidligt, kan fejl findes tæt på tidspunktet, hvor de begås. Ved at være opmærksom på dem fra start, kan man endda forhindre, at nogle af dem opstår.

Der findes på denne måde mange fejl i starten af testen. Det skærper opmærksomheden og understreger behovet for tid og ressourcer til test.

9. Fejlanalyse

Fejlanalysens formål er at:

1) Forhindre at fejlene begås en anden gang ved at pege på årsagen, og definere de nødvendige præventive tiltag.

2) Måle produktivitet. Der indgår altid en fejltælling (Defect Rate) i de anerkendte måder at måle produktivitet på. Enten ved at tælle fejl pr. tusind linier kode, fejl pr. side kravspecifikation eller fejl pr. Function Point. Enhver kan præstere en høj produktivitet, hvis antallet af fejl er ligegyldigt.

Den systematiske fejlanalyse bruges mest af forvaltere. Det er på de fejl, der findes i produktion, og dermed som oftest er blandt de dyreste. Det er måske ikke kun koden der skal repareres, men også data.

Men der er fordele ved også at analysere fejl, der findes i udviklingsforløbet. Det er i hele træskolængder de samme typer, der risikerer at slippe igennem til drift. Og det er værd at finde årsagerne til dem med det samme.

Hvis vi låner en fejlanalyse fra systemforvaltningen, foregår den ved hjælp af spørgsmålene figur i 2.

Hvem opdagede fejlen?
Hvordan blev den opdaget?
Årsagen til fejlen?
Hvornår blev fejlen begået?
Hvem begik fejlen?
Hvordan kunne fejlen være undgået?
Hvordan kunne fejlen være fundet tidligere?
Hvad er konsekvenserne af fejlen?
Hvad er de samlede omkostninger?

Figur 2: Gode spørgsmål i en fejlanalyse

Fejlrapporterne registreres i en database, og når indrapporteringen af hver af dem er gjort præcist, kan fejlanalysen foregå maskinelt. Den kan give information om:

- et bestemt projekt
- på tværs af alle udviklingsprojekter det år
- for alle projekter i en given periode
- for enkelte produkter i forvaltning
- for alle produkter i forvaltning i en given periode, etc.

Kun behovet sætter grænser. Analysen kan dokumenteres i søjler og lagkager, der peger direkte på, hvor der er penge at spare.

10. Afsluttende bemærkning

Argumenterne for at lade være med at begå fejl, er bevidst trukket frem. Der er ingen tvivl om at det er billigere at bruge tid på at undgå fejl, end at finde og reparere dem. Kan man udføre en reparation så omhyggeligt, som det er nødvendigt. Teknisk kan man godt, men er der tid til det?

Konklusionen er også, at hvis programmører kunne nøjes med at udvikle, inklusive at debugge egne fejl, ville flere projekter lande til tiden. Den store mængde spildtid opstår når de skal rette fejl der stammer fra deres arbejdsgrundlag, for eksempel kravspecifikation eller Use Cases.

P.S.

Hvad var svaret på spørgsmålet om de 101,- kr.? Var det en fejl i uddata, fejl i to moduler eller en specifikationsfejl? Spørgsmålet er relevant, hvis der arbejdes seriøst med fejlanalyse. I modsat fald har det ingen betydning. Men hvis vi ønsker at lære af vores fejl for at definere præventive tiltag, skal fejlen registreres som specifikationsfejl. Den har kostet tusinder af kroner at reparere, men burde kun have kostet fem. Det kan altid diskuteres om krav skal dokumenteres på det niveau, men hvis der er ønske om 0-fejl, så er diskussionen i sig selv et godt skridt på vejen.

Kvikstart

Vedligehold en database over fejl. Analysér dem jævnligt, og som minimum med hensyn til deres egentlige årsag, omkostninger og hvilke midler der skal bruges for at undgå dem fremover. Udvid løbende analysen til at omfatte alle de spørgsmål der er nævnt i figur 2. Fejl der opleves af kunder skal behandles som en sag der vedkommer alle.

Det er ledelsens ansvar, at der udføres en pålidelig fejlanalyse. Brug analysen til at mindske omkostningerne til fejl. Men formulér ikke et mål for fejlfrihed der skal opnås på et bestemt tidspunkt. Formulér hellere at omkostningerne ved fejl skal falde med en bestemt procentsats hvert år.

Tilbage til toppen af tekst