Engineer

Programvarefeil

en feil som hindrer et produkt i å oppfylle kravene sine

Hva det er

En programvarefeil er noe galt inne i selve arbeidet, som får det til å oppføre seg annerledes enn tiltenkt eller til å utelate noe det skulle ha gjort. I hverdagen sier de fleste utviklere bug eller bare feil, i testfaget heter det ofte defekt, og engelske standarder skriver også fault og flaw.

Navnet sier programvare, men det viktige ordet er *arbeidet*. En programvarefeil kan ligge i kildekoden, og like gjerne i et krav som motsier et annet krav, et design som ikke tåler lasten det ble tegnet for, en konfigurasjonsfil, en test som forventer feil verdi, eller en brukerveiledning som beskriver en knapp produktet fjernet for to versjoner siden. Uansett hvor den ligger, hører feilen til selve tingen, og den er der enten noen har lagt merke til den eller ikke.

Testfaget holder tre nabobegreper fra hverandre, og skillet betaler for seg. ISTQB bruker de engelske ordene error, defect og failure; her heter de feilgrep, programvarefeil og svikt:

- **Feilgrepet** gjør et menneske: leser spesifikasjonen galt, antar noe som ikke stemmer, eller treffer feil tast.
- **Programvarefeilen** er det feilgrepet etterlater i det personen lagde.
- **Svikten** er den synlige, gale oppførselen som kommer når den defekte delen kjører under forhold som avslører den.

Kjeden går bare én vei og kan stoppe ved hvert ledd. En programvarefeil i en gren som aldri kjøres, eller en som trenger inndata som ingen sender, kan ligge i en kodebase i årevis uten å gi én eneste svikt.

Hva som regnes som galt, avhenger av hva som var forventet, og forventningene rekker lenger enn den skrevne spesifikasjonen. Produktkvalitetsmodellen i ISO/IEC 25010 beskriver funksjonell egnethet som i hvilken grad et produkt leverer funksjoner som dekker uttalte og underforståtte behov når det brukes under bestemte forhold. Modellen deler egenskapen i tre:

- **Fullstendighet:** dekker funksjonene de spesifiserte oppgavene og målene til dem de er laget for?
- **Korrekthet:** er resultatene riktige?
- **Hensiktsmessighet:** hjelper funksjonene brukeren å få oppgaven gjort?

En gal sum er et korrekthetsproblem; en rapport som ikke lar seg eksportere, er et fullstendighetsproblem. Den samme modellen behandler pålitelighet som en egen egenskap, med bl.a. feilfrihet, tilgjengelighet, feiltoleranse og gjenopprettbarhet som deler. Et produkt som regner ut riktig svar og så krasjer to ganger om dagen, har altså fortsatt programvarefeil.

En regresjon er en programvarefeil med historie: noe som virket, og som en senere endring ødela. Den har fått sitt eget navn fordi den har med seg et spor som en opprinnelig feil mangler, nemlig en siste versjon som er kjent for å være god.

Bruk begrepet i en beslutning

Bruk «Hva er en programvarefeil?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Ett team som tar en idé fra skisse til noe kundene dine kan logge inn på, i stedet for at den sendes videre mellom tre leverandører.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Ledet en GIS-kartapp til 10 ganger mer omsetning — 10x inntektsvekst, ledet fra kartproduktet

Ledet en GIS-kartapp til 10 ganger mer omsetning Ta kontakt →

Hvorfor det er viktig

Hva en programvarefeil koster, avhenger mest av hvor den blir funnet. Fanger den som skriver koden feilen selv, koster den et minutt og etterlater ingen spor. Fanges den i kodegjennomgang, koster den en samtale. Finner en kunde den, koster den selve svikten, saken hos kundestøtte, undersøkelsen, rettelsen og litt tillit, og har den gjort reell skade, er den blitt en del av en driftshendelse.

Typesystemer, lintere, tester og kodegjennomgang jobber alle på samme linje: de prøver å flytte øyeblikket feilen blir oppdaget, nærmere øyeblikket den ble skrevet.

Programvarefeil kommer dessuten i familier. Feilgrepet bak én har som regel slektninger: det samme misforståtte grensesnittet kalt på samme måte seks steder, den samme forvekslingen av enheter kopiert inn i hver ny modul, den samme usjekkede antakelsen i hver forespørselshåndterer. Etter en feil er det nyttigste spørsmålet ikke så mye hvor den ødelagte linjen står, som hva som gjorde linjen lett å skrive, og hvor det samme har skjedd andre steder.

Og programvarefeil kommer tilbake. Koden endres hele tiden, og en rettet feil kan vende tilbake gjennom en refaktorering, en merge eller en oppgradering av en avhengighet. En test som er skrevet for å feile på feilen og passere på rettelsen, gjør rettelsen til en varig egenskap ved bygget. Dukker en regresjon opp likevel, blir versjonshistorikken et søkerom. `git bisect` gjør et binærsøk over commitene mellom en kjent god og en kjent dårlig versjon, så et spenn på tusen commits snevres inn til den ene som innførte feilen, på omtrent ti testkjøringer.

Noen programvarefeil er også sikkerhetsproblemer. MITREs Common Weakness Enumeration, CWE, katalogiserer svakheter, som den beskriver som tilstander i en komponent i programvare, fastvare, maskinvare eller en tjeneste som under visse omstendigheter kan bidra til at det oppstår sårbarheter. Mange oppføringer på den listen er helt vanlige programvarefeil som en angriper har bruk for: en manglende grensesjekk, brukerinndata sendt uescapet inn i en spørring, et autorisasjonstrinn som hoppes over på én rute.

Slik ser det ut i praksis

Programvarefeilene som kostet mest på dette nettstedet, var de som alle automatiske signaler meldte var i orden. En gjennomgang avdekket elleve feil i universell utforming som de automatiske reglene hadde godkjent, fordi hver av dem var en egenskap ved hvordan en side ble gjengitt, og ikke ved hva oppmerkingen sa. To av dem satt i selve kontrollene. Hvert vindu layoutkontrollen åpnet, var 900 piksler høyt, så en telefon i liggende format på 852 ganger 393, med sju piksler mellom to elementer, var aldri blitt sett på; og regelmotoren hadde kjørt 70 av sine 105 regler. Hver feil gikk inn i et register

sammen med regelen den ga opphav til, slik at arbeidet lukker en feilklasse og ikke bare ett tilfelle. Da layoutkontrollen selv var rettet, feilet den 128 ganger på tvers av tre språk.

Den neste feilen var helt uten symptomer. Nettstedet har et mørkt tema som standard og et lyst tema for lesere der systemet ber om det, og i det mørke temaet er dekket som fyller hele siden, bygd av tre lag. En liten strukturell kontroll som teller lagene hvert tema maler, fant at det lyse temaet bare tegnet to allerede ved første kjøring. Fasettoverlegget hadde manglet siden det lyse temaet ble skrevet. Hver side ble gjengitt uten én eneste feilmelding, og en bakgrunn med ett lag mindre ser ut som et bevisst valg, så i månedsvis var det ingen grunn til å se etter. Rettelsen var én regel; det er kontrollen som holder den på plass.

Å holde rettelser på plass er jobben til repoets kontrollmotor, der én regel knytter hver feilretting til bevis: en commit merket som rettelse må ha med en test som feiler uten den. Merket er prefikset `fix:` . Kontrollene har i tillegg 146 egne tester, fordi en kontroll med en feil kan slippe gjennom endringene den skulle avvise, og se ut til å virke.

Hvor det går galt

Tilfellet blir rettet, og feilklassen står åpen. Den rapporterte linjen blir rettet, saken lukkes, og tre søsken av det samme feilgrepet venter andre steder i koden. Søk etter mønsteret som ga feilen etter hver rettelse, og der en maskin kan kjenne igjen mønsteret, gjør søket om til en kontroll.

En rettet feil kommer tilbake. En refaktorering et halvt år senere gjeninnfører den gamle oppførselen, fordi ingenting registrerte hva rettelsen beskyttet mot. Skriv testen før rettelsen, se den feile på den gamle koden, og behold den i testsuiten for godt.

Kontrollen bærer selv feilen. Et regelsett som i det stille laster en brøkdel av reglene sine, en liste over testvinduer som alle har samme høyde, en assert som sjekker hva koden gjør i stedet for hva den skal gjøre: hver av dem melder suksess mens feilen er åpent synlig. Gi kontrollene egne tester, og les i verktøyets egen utdata hvilke regler som faktisk kjørte.

Testdekning tar plassen til korrekthet. En testsuite kan kjøre hver eneste linje og likevel godta et galt resultat når ingen assert ser på utfallet, og kode som følger den skrevne spesifikasjonen, kan fortsatt bomme på et underforstått behov. Bruk testdekning til å finne kode som ingenting tester, og vurder en test etter hvilke feil den ville fanget.

«Fungerer som designet» lukker en reell rapport. Oppførselen stemmer med spesifikasjonen, rapporten avvises, og det defekte arbeidsproduktet var spesifikasjonen hele tiden. Når problemet ligger i kravet eller designet, registrer feilen mot det, slik at rettelsen skjer ved kilden.

Les mer

- Defect — ISTQBs definisjon fra programvaretesting, med synonymene og de beslektede begrepene error og failure.
- ISO/IEC 25010 — en oversikt over produktkvalitetsmodellen og underegenskapene til hver egenskap, bl.a. funksjonell egnethet og pålitelighet.
- git-bisect — Gits manual for binærsøket som finner commiten som innførte en feil, også automatiserte kjøringer med et testskript.
- About CWE — hva Common Weakness Enumeration er, hvordan listen over svakheter i programvare og maskinvare er organisert, og hvem som driver den.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en programvarefeil?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/defect/