VIL VI NOEN GANG FÅ FEILFRI PROGRAMVARE?

Feil i programvare har i senere tid flere ganger ført til krisestemning og svekket tillit til IT-støttede tjenester. To nærliggende eksempler er sikkerhetsproblemene med skatteetatens Altinn og signaleringsfeilene som førte til mobilnett-kollaps for Telenor. Det er ikke første gang feil i IT-systemer har medført problemer og dette er heller ikke unikt for Norge. I 2002 konkluderte en studie i USA med at det var et innsparingspotensial på ca. 0.6% av bruttonasjonalproduktet ved å oppdage feil tidligere og å unngå programvarefeil i leverte IT-systemer (www.nist.gov/director/prog-ofc/report02-3.pdf). Studien har mange svakheter, men gir kanskje likevel et bilde av størrelsesordenen til de direkte kostnadene ved programvarefeil. Antar vi at Norges bruttonasjonalprodukt er på ca. 2700 milliarder kroner (www.ssb.no/regnskap), så utgjør 0.6% et årlig innsparingspotensial på ca. 16 milliarder kroner - ikke akkurat småpenger det heller. Hvor realistisk er imidlertid et mål om feilfri programvare?

At mennesker gjør feil, små og store, som fører til at ting ikke virker som de skal og har problemer med å oppdage dem, er ikke noe nytt og ikke spesifikt for programvare. Begrepet ”bug”, som ofte brukes om programvarefeil, er i så måte illustrerende. Allerede i 1878 ble dette begrepet brukt av Thomas Alva Edison for å beskrive hans opplevelse av at små feil ofte førte til ting tok mer tid å lage enn forventet. Æren for å introdusere begrepet ”bug” i programvaresammenhenger gis ofte, trolig feilaktig, til Grace Hopper og hennes team ved Harvard. I teamets loggbok fra 1947 finner vi bruken av ”bug” å beskrive en møll som hadde forårsaket feil i utregningene på datamaskinen Mark II. Møllen var for øvrig limt inn i loggboken og siden er nå utstilt på ”National Museum of American History”.

Det er gode grunner til at det oppstår feil i IT-systemer. Et typisk IT-system i dag består av flere hundre tusen, ofte millioner, instruksjoner. Disse instruksjonene er skrevet av ulike programmerere i ulike team og miljøer, og henger sammen på utallige måter. Noen av sammenhengene er det svært vanskelig, for ikke å si umulig, å ha full oversikt over. Det er tilstrekkelig med små feil, som et feil plassert komma, og programvaren kan feile katastrofalt. Ikke nok med at ”liten tue kan velte stort lass”, men programvaren skal typisk virke med mange ulike teknologier og på måter som det er vanskelig å forutse for programmererne. Resultater fra en studien: ”Common trends in Software faults and failure data”, IEEE Transactions on Software Engineering, 2009, illustrerer dette. Basert på et stort åpen kildekode-system og et enda større system utviklet av NASA, gjorde forskerne flere interessante funn. De fant at hele 60% av feilene var sammensatte/spredte og at korrigering krevde forståelse og oppdatering av flere filer. Så mange som 30% omfattet mer enn en komponent, gjerne utviklet av andre leverandører. Dette illustrerer at man må teste kombinasjoner av tilstander til ulike komponenter for å avdekke alle feil. Når antall komponenter øker, vil dermed antall kombinasjoner fort bli astronomisk høyt og en

fullstendig test vil i praksis være umulig for store, komplekse systemer. Forskerne fant også at Pareto-prinsippet i høy grad var gyldig for programvarefeil. Ca. 20% av programvaren hadde 80% av feilene. Ordtaket om at ”en ulykke kommer sjelden alene” gjelder altså også for programvare. Kildene til feilene var av ulike typer, og kun en tredjedel av dem var relatert til programmeringsfeil. Resten var relatert til feil i kravene, feil i data-ene, feil i utforming av design, feil i integrering med andre systemer osv. Selv om man – som mange tidligere håpet på – skulle lykkes i å bevise at programvaren var korrekt i forhold til en spesifikasjon, så ville man dermed kun være i stand til å fjerne en mindre del av alle programvarefeil. Formell verifikasjon av programvare er dermed ikke bare en relativt kostbar måte å drive programvareutvikling, men har også store begrensninger i virkeområde.

Det synes klart at vi neppe klarer å unngå å introdusere feil i programvare. Det vil fortsatt være menneskelig å feile, særlig når kompleksiteten blir høy. Vi vil neppe heller i nær fremtid vil være i stand til å bevise at programvare er korrekt, all den tid dette forutsetter feilfri og fullstendig spesifikasjon. Hva så med mulighetene for å fjerne det meste av feil ved å teste bedre?

God testing er essensielt for IT-systemer av høy kvalitet. Som studien ovenfor indikerer er det imidlertid urealistisk å teste alle relevante kombinasjoner og bruksområder for programvare. Det er derfor sjelden at bedre testing vil kunne fjerne alle feil. I 2002-studien fra USA ble utviklerne bedt om å anslå hvor mye det var realistisk å kunne forbedre med bedre prosedyrer og metoder for test. Svarene tilsa at de trodde det var realistisk at bedre testing kunne halvere kostnadene forbundet med programvarefeil. Det de ikke ble spurt om, og som ser ut som en svakhet med undersøkelsen, var hvilken kostnadsside dette ville ha. Det å finne feil tidligere, finne flere feil og å være sikrere på at programvaren som leveres ikke har feil medfører typisk ekstrakostnader. Det å ha som mål å fjerne de siste feilene – noe man for øvrig aldri vil vite om man har gjort - vil for eksempel kunne medføre at utviklingstid og kostnader blir uhensiktsmessig høye. I praksis er det dermed ofte kunden som bestemmer kvaliteten på systemet ved å sette krav til og å være villig til å betale for omfattende testing av et IT-system. En av utviklerne sier det slik i ”Why do programmers make security errors?” (IEEE Symposium on Visual Languages and Human-Centric Computing, 2011): ”*If (the) customer cares about security then the company has to care about security.*”

Oppsummert tyder forskning og erfaring på at vi neppe noen gang vil få feilfri programvare og at færre feil i IT-systemer i stor grad er et økonomisk spørsmål om kost og nytte. Det er dermed bare å innstille seg på at IT-systemene vil feile, noen ganger katastrofalt, også i framtiden.