Inżynieria programowania

Dr Janusz GWIAZDA

Do niedawna programiści uważali siebie i swoją profesję jeśli nawet nie za coś w rodzaju powołania, to na pewno za rodzaj sztuki. Tym szlachetniejszej, że wywodzącej się nieomal wprost z matematyki. Sztukę konstruowania programów uprawiano z mniejszym lub większym powodzeniem, ale zawsze z zamglonymi oczyma i rozwichrzoną czupryną.

Próby traktowania informatyki po prostu jako działu matematyki były obiecujące. W tym też kierunku skłaniali się ci, którzy w rozwoju metod numerycznych, teorii automatów, algorytmów itp., upatrywali rozwiązania problemu (przed którym stawał zawsze każdy autor programu) jak upewnić się, że zbudowany program jest już poprawny.

Inni, którym matematyka nie była tak bliska, skłonni byli raczej poszukiwać praktycznych metod weryfikacji programów, sposobów praktycznego postępowania eliminującego błędy.

Jedni i drudzy wiedzieli już bowiem, że rzeczą ludzką jest mylić się i programy z niewykrytymi błędami są zjawiskiem naturalnym.

Po zastanawiająco krótkim, bo dziesięcioletnim okresie sporów między tymi skrajnymi tendencjami, upływ czasu i pewna konieczność (o której dalej wspomnimy) rozstrzygnęły już chyba, że programowanie, rzetelne programowanie, jest mieszaniną szczypty artystycznego natchnienia i wielu, wielu garści dobrego rzemiosła. Nic w tym nowego — w każdej dziedzinie ludzkiej działalności da się te dwa elementy odnaleźć, co najwyżej proporcje mieszaniny są inne. Spory zresztą wcale nie były akademickie, miały bowiem istotne znaczenie praktyczne nie tylko dla technik programowania i związanych z nimi wszelakich rutyn organizacyjnych, lecz także dla sposobów kształcenia informatyków, doboru ludzi do zawodu itp.

Rozwijano niemal równolegle, choć niezależnie, dwie drogi: jedna z nich to mozolne poszukiwanie metod formalnego (a więc tak beznamiętnego jak tylko matematyka potrafi) dowodzenia poprawności programów; druga droga to gromadzenie (nawarstwiającej się w w wyniku obserwacji i doświadczeń) wiedzy o sposobach pracy programistów i technikach organizatorskich doskonalących tę pracę.

Pojawienie się mikroprocesorów postawiło problem niezawodności programów w nowym, dramatycznym świetle. Okazało się bowiem, że dowodzenie poprawności programów może się po prostu opłacać.

Do tej pory dowody (znacznie dłuższe niż sam program i od niego zwykle mniej zrozumiałe) budziły raczej wesołość praktyków, jako nieszkodliwe dziwactwa matematyków. Im samym było zresztą również nieswojo, że wymyślają coś tak niepraktycznego. Mikroprocesor jednak, któremu dla pewnych zastosowań wbudowuje się program na stałe (np. dla sterowania pralką) może być — i często jest — powielany w kilkuset tysiącach egzemplarzy. Wyobraźmy sobie, że program na stałe wbudowany w setki tysięcy pralek ma błąd — plajta firmy jest murowana. Okazuje się więc, że dla takich zastosowań formalny dowód poprawności programu (mimo iż znacznie kosztowniejszy od samego programu) jest nie tylko opłacalny, ale i konieczny.

Czy wobec tego już dziś każdy program dla komputera ma również dowód swojej poprawności? Wręcz przeciwnie. Po pierwsze: sposoby dowodzenia są jeszcze zbyt kosztowne. Po drugie: nie tak znowu wielu ludzi umie to robić. A wreszcie: nie zawsze przecież warto to robić (budując, powiedzmy, płot wokół działki nie przeprowadza się obliczeń statycznych, a przy kopaniu studni też raczej korzysta się z różdżkarza, niż z badań geologicznych i jakoś najczęściej wodę się ma).

Co więcej, sama poprawność nie wystarczy. Potrzebne jest jeszcze coś. W poważniejszych, kosztownych przedsięwzięciach informatycznych wymaga się obecnie, przy budowaniu oprogramowania, stosowania standardowych technik postępowania i dokumentowania pracy — to zaś wszystko razem (łącznie z dowodami niezawodności elementów) nie jest niczym innym jak inżynierskim postępowaniem. Jednym z zaleceń inżynierii (każdej, a nie jedynie programowania) jest wymaganie precyzji przy opisie — dokumentowaniu powstałej konstrukcji.

Programiści (jeszcze w znacznej liczbie artyści) dokumentacji programów nie znosili, uważając je za nużącą i całkowicie zbędną biurokrację. Na ogół też udawało się im (i udaje) w mniejszym czy większym stopniu od tego wykręcać. Nauczyciele programistów, mówiący o potrzebie dokumentowania, też raczej robili to z przekonania o potrzebie dobrych manier, niż absolutnej konieczności.

Zdarzają się jednak przypadki, które wskazują na konieczność inżynierskiego (także rutynowego, a nie tylko z polotem) postępowania przy konstruowaniu programów. Jeden z takich przypadków miał miejsce w wojnie o Falklandy (czy Malwiny — jak kto woli). W eskadrze brytyjskiej mianowicie płynęły dwa supernowoczesne niszczyciele. Okręty, o których mówiono, że bronione są niezawodnie nie tylko masą urządzeń strzelających, lecz także że sterowanie tym całym precyzyjnym żelastwem przez komputery przewyższa wszystko, co mogliby zrobić ludzie, jest bowiem niezawodne — poprawne. Jakiż był więc szok, gdy w pierwszym starciu z lotnictwem zrzucającym torpedy okazało się, że jeden z niszczycieli został trafiony i zatonął, drugi zaś, poważnie uszkodzony, również przestał uczestniczyć w walce.

Analiza przyczyn takiej klęski (na szczęście zresztą nie ukrywana) pokazała na tyle dobitnie czym jest brak pełnego inżynierskiego postępowania przy budowie programów sterujących obroną, że nawet nieskłonna do wydatków z budżetu Pani Thatcher wyasygnowała kilkaset milionów funtów dla brytyjskich fakultetów kształcących informatyków. Okazało się bowiem, że obrona przeciwtorpedowa, a raczej program nią sterujący uważał zbliżającą się torpedę za tzw. przyjacielską (torpedy były francuskie) i statek nie reagował ani manewrami, ani próbami zniszczenia torpedy.

Co ma do tego inżynieria oprogramowania? Otóż ma. Obsługa statku, jeszcze przed starciem z lotnictwem, wiedziała o ewentualności użycia „przyjacielskich" torped. Nie była jednak w stanie zmodyfikować programu sterującego obroną — był skonstruowany „artystycznie", jego dokumentacja prawdopodobnie też.

Zazwyczaj programiści, otrzymując zadanie poprawienia lub zmodyfikowania nie przez siebie napisanego programu, proponują napisanie go od nowa. Powodem jest olbrzymia trudność w rozszyfrowaniu konstrukcji — najczęściej niestety (jak już było powiedziane) pobieżnie lub wręcz błędnie opisanej.

Pewnym usprawiedliwieniem zadziwiającego, wydawałoby się, zjawiska, jakim jest niechęć do dokumentowania (w tak jeszcze świeżej dziedzinie jak informatyka), niech będzie fakt budzący moje szczere zdumienie, że i w innych, zdawałoby się inżyniersko ugruntowanych dziedzinach zdarza się to wcale nierzadko. Ot, choćby stałe kłopoty z wykopkami ulicznymi przy budowie arterii komunikacyjnych — nikt nie wie dokładnie, gdzie podczas kopania natrafi na kable, przewody gazowe czy kanalizacyjne.

Prawa Murphyego w informatyce

Jeżeli cokolwiek może być zrobione źle, to na pewno będzie.

Każdy sprawdzony i działający program zawiera przynajmniej jeden błąd.

Najwcześniej w pół roku od rozpoczęcia użytkowania programu zostaje w nim wykryty najpoważniejszy błąd.

Liczba błędów niewykrywalnych jest nieskończona, w przeciwieństwie do, z definicji ograniczonej, liczby błędów wykrywalnych.

Najpotrzebniejsze dane są zawsze najtrudniej dostępne.

Złożoność programu rośnie, aż przekroczy możliwości programisty, który go używa.

Zadania

Redaguje mgr Krzysztof S. NOWIŃSKI

M 362. Wykazać, że dowolna potęga liczby 376 kończy się cyframi 376.
Rozwiązanie na str. 6

M 363. Każda z $2n$ osób na zebraniu towarzyskim ma wśród obecnych najwyżej $n-1$ wrogów. Czy można posadzić wszystkich za okrągłym stołem tak, by nikt nie siedział obok swego wroga?
Rozwiązanie na str. 3

M 364. Odległość między punktami A i B na płaszczyźnie wynosi 1. Zbudować taki kwadrat o bokach zawierających A i B, by suma odległości od A do wierzchołków tego kwadratu była najmniejsza z możliwych.
Rozwiązanie na str. 11

Redaguje mgr Tomasz TRATKIEWICZ

F 152. Z jaką minimalną prędkością v względem Ziemi należy wystrzelić sztucznego satelitę, aby opuścił Układ Słoneczny?
Rozwiązanie na str. 11