**Komputer dan Kecelakaan Penerbangan**

**“Roket Ariane 5”**

Ledakan bunga api yang mengakhiri penerbangan roket Ariane-5 pada tanggal 4 Juni 1996, mendorong kita untuk lebih memikirkan permasalahan atau bahaya yang diakibatkan kesalahan program. Roket Ariane ini setelah 40 detik berbelok dari arah yang direncanakannya, terpaksa diledakkan dan dihancurkan dengan alasan keselamatan (*safety*). Barang bawaan berupa 4 buah satelit juga terpaksa dihancurkan dan mengakibatkan kerugian lebih dari 600 juta dollar US.

Kesalahan pemrograman.

Pada awalnya kecelakaan Ariane ini hanyalah dianggap seperti kesalahan pemrograman biasa atau yang dikenal dengan istilah "bug". Program untuk pengendalian Ariane ini ditulis dengan menggunakan bahasa pemrograman Ada, suatu bahasa pemrograman yang tergolong aman. Program kendali ini dijalankan pada komputer on-board. Komputer ini melakukan konversi data 64 bit floating point ke 16 bit integer, yang ternyata tidak selalu berhasil dengan baik. Pada saat terjadi kecelakaan nilai hasil konversi ini menjadi terlalu besar, maka konversi dianggap mengalami kegagalan. Hal ini menjadikan program melakukan proses penanganan kesalahan. Bersamaan dengan hal ini sesuai dengan aturan, maka kedua komputer akan segera dimatikan. Sehingga tak ada lagi informasi untuk sistem navigasi dan terjadilah bencana tersebut.

Tetapi bug tersebut bukanlah sekedar kesalahan pemrograman. Komputer navigasi memiliki rutin perangkat lunak yang diambil dari model pendahulunya, Ariane 4. Roket ini memiliki perilaku yang belum pernah dibandingkan dengan kondisi Ariane 5, atau telah diuji dengan seksama dengan komponen Ariane-5 [1]. Kontradiksi antara perilaku yang valid pada Ariane 5 dan persyaratan dari Ariane 5 telah tertimbun jauh dalam dokumentasi pembuatan Ariane. Bahkan proses pengembangan yang telitipun tak mampu menguak hal tersebut.

Sistem untuk misi amat penting (*mission-critical system*) sudah sepatutnya diprogram sehingga mereka dapat menangani kondisi ketika memiliki nilai masukan atau keluaran yang salah. Sudah barang tentu variabel lainnya pada program yang sama di Ariane telah dilindungi dari terjadinya kondisi "*overflow"* ini. Tetapi dalam pertimbangan agar tak mengakibatkan beban kerja prosesor yang terlalu berat, maka proses perlindungan ini hanya dilakukan pada variabel-variabel yang sangat penting. Jelas pada kondisi ini programmer tak dapat disalahkan begitu saja Sebab sebelum mereka memutuskan untuk tak melindungi variabel tersebut, mereka telah membuktikan secara matematis bahwa tak ada nilai salah yang dapat timbul pada variabel yang tak terproteksi tersebut. Akan tetapi, pembuktian matematis untuk variabel tak terproteksi tersebut hanya valid untuk pola jalur lintas Ariane 4 yang lama.

Bila kita mempertimbangkan proses proteksi kisasan nilai ini, pertanyaan akan mengarah kepada pertimbangan, mengapa uji kisaran nilai ini dan pengujian lain tidak merupakan suatu kuajiban ketika digunakan pada perangkat lunak pada sistem tertentu ? Pertanyaan ini adalah pertanyaa terbuka dan tak memilki titik akhir jawaban. Sebagai ilustrasi akan dikemukakan salah satu contoh yang menunjukkan bahwa pada suatu sustem pengujian "safety" seperti ini bahkan menimbulkan suatu permasalahan tambahan tersendiri. Pada 15 Januari 1990, perangkat switch perusahaan AT&T yang berlokasi di pantai timur Amerika mengalami kelebihan beban ( *overload*), sehingga gagal melaksanakan tugasnya. Suatu modul untuk perbaikan pada saat itu melakukan inisialisasi ulang node tersebut pada suatu keadaan yang inkonsisten dan kemudian melakukan start ulang. Kondisi ini menghasilkan reaksi berantai yang mengakibatkan prosedur reboot di seluruh sistem telekomunikasi.

Mekanisme perbaikan (*recovery mechanism*) juga merupakan suatu perangkat lunak, yang merupakan bagian dari seluruh sistem. Keseluruhan perangkat lunak ini harus diimplementasikan secara benar. Sehingga penambahan modul perbaikan ini juga akan memberikan suatu permasalah tambahan. Semakin banyak mekanisme perbaikan yang disertakan pada suatu sistem semakin besar pula kemungkinan kesalahan implementasi mekanisme tersebut.

**Solusi, masalah dan jenis kesalahan**

Pada umumnya, setiap solusi akan menimbulkan permasalahan baru bila diterapkan secara berlebihan. Yang menjadi pertanyaan adalah dimanakah batasan antara "normal" dan "berlebihan". Pada tahun 1980-an sosilog Charles Perrow melontarkan argumen bahwa pada suatu sistem yang kompleks kesalahan seperti ini memang tak dapat dihindarkan [3]. Menurut Perrow, kesalahan sistem seperti ini cenderung terjadi pada sistem dengan kondisi di bawah ini

* Sistem terkait erat *(tightly coupled)*. Suatu perubahan kecil akan menjalar dan segera memberikan dampak pada sistem secara keseluruhan
* memiliki banyak "agen pelaksana tugas" berupa prosesor atau manusia yang berkomunikasi satu sama lain secara intensif secara kompleks.

Pada kasus Ariane di atas, kesalahan juga terletak pada definisi kebutuhan (*software requirement*) perangkat lunak dan bukan pada jenis kesalahan pada sistem di atas. Kecelakaan seperti di atas dapat juga terjadi pada sistem yang kecil. Penggunaan ulang perangkat lunak ( *software reuse*) hanya dapat sukses setelah melalui analisis kebutuhan yang seksama. Kasus Ariane telah menunjukkan sejauh mana suatu analisis kebutuhan perangkat lunak seharusnya dilaksanakan. Kesalahan kisaran nilai hanyalah sebuah symptom bukan penyebab

Suatu studi di NASA Jet Propulsion Laboratory telah menunjukkan bahwa lebih dari 95% dari seluruh masalah pada sistem amat penting (mission-criticial) berada pada tingkat penentuan kebutuhan (requirement level) bukan pada disain pemrograman [5]. Requirement atau penentuan kebututhan berarti menentukan "*Apa yang seharusnya dilakukan suatu sistem*? *Apa kebutuhan yang harus dipenuhi oleh sistem*". Sedangkan disain berarti menentukan "*Bagaimana suatu sistem seharusnya memenuhi kebutuhan tersebut* ? Dan bisa dipaparkan baik dalam terminologi umum (spesifikasi disain) atau dalam bentuk detail (sebagai source code pada bahasa tingkat tinggi).

Kesalahan suatu sistem dapat diklasifkasikan menjadi tiga kategori :

* **Kesalahan murni perangkat keras**, misal kerusakan transistor. Kesalahan ini hanya dapat didiagnosis ketika terjadi. Perbaikan dilakukan dengan penggantian perangkat keras, misal dengan mengganti microchip.
* **Kesalahan tingkat menengah** antara konsepsi ke disain perangkat keras, misal Pentium bug. Secara teoritis kesalahan ini dapat diketahui sebelum terjadi dan dihindari. Hal ini disebab setiap evel dari kesalahan ini dapat dibandingkan secara matemtis dengan tingkat lebih tinggi, dalam upaya membuktikan bahwa tingkat lebih rendah mengimplementasi tingkat yang lebih tinggi secara benar. Metoda formal menggunakan pendekatan ini
* **Kesalahan requirement.**Sangat berbeda dengan tingkat menengah, requirement atau paparan kebutuhan tak dapat diverifikasi secara matematis. Apa yang kurang atau tak tepat pada tingkat abstraksi yang lebih tinggi tersebut, tingkat kebenarannya (*correctness*) tak dapat diturunkan. Kesalahan requirement hanya dapat ditemukan pada pengujian menyeluruh (*all-up test*) atau dengan membandingkan spesifikasi dari formalisasi dari dua formalisasi yang terpisah atau intuisi, paling tidak dapat dibuktikan pada sebagian dari keseluruhan sistem.

Permasalahan "pembalikan prioritas" (*priority inversion*) pada Mars pathfinder, yang menyebabkan kondisi reset dan penundaan pelaksanaan misi merupakan salah satu bentuk permasalah tingkat menengah walaupun juga relatif merupakan tingkat tinggi [6]. Kesalahan tingkat menengah sederhana seperti penanganan MIME pada Microsoft Outlook atau Netscape Messenger (yang juga merupakan permasalahan kisaran nilai, juga merupakan kesalahan pemrograman sederhana) [7] sangat jarang ditemukan pada sistem yang ditujukan untuk misi sangat penting. Pada sistem sangat penting source code akan diperiksa dengan amat sangat seksama oleh tim pengembang perangkat lunak, bahkan dimodifikasi berulang kali. Oleh sebab itu mengapa NASA lebih menyukai sistem operasi seperti Linux atau vxWorks, karena ketersediaan source code dari perangkat lunak, sehingga memungkinkan untuk dianalisis dan dimodifikasi sesuai dengan kebutuhan misi yang sangat penting tersebut. ibuktikan pada sebagian dari sistem keseluruhan.

**Komputer dan pesawat terbang**

Suatu pesawat terbang agar dapat berangkat, terbang dan mendarat dengan selamat, maka orang, peralatan mekanik dan elektronik di pesawat terbang, juga mekanisme seluruh sistem pengaturan lalu lintas udara harus melakukan kerja sama sebagai suatu sistem heterogen. Komputer pada pesawat terbang merupakan bagian dari sistem keselurhan tersebut. Dalam hal ini kesalahan pemrograman belum tentu dapat dituding langsung sebagai penyebab sebuah kecelakaan. Tetapi suatu kecelakaan atau kejadian (*incident*dan *accident*) tak hanya dapat timbul oleh kesalahan pada salah satu komponen sistem, tetapi juga dari pelaksanaan pekerjaan yang mengkombinasikan keduanya.

Hampir selalu terjadi beberapa faktor yang tak dikehendaki terjadi secara bersamaan seperti pendaratan pesawat Airbus A320 di Warsawa pada September 1993 yang menyebabkan 2 orang meninggal [4]. A 320 adalah pesawat komersial pertama yang menggunakan peralatan kemudi terkontrol komputer yang tak dapat diambil alih oleh pilot (*fly by wire*). Hal ini dilakukan untuk melakukan pengurangan bobot pesawat serta meningkatkan kehandalan perangkat keras, dan mencegah dari kesalahan manusia dan beberapa hal lainnya. Pesawat ini dikemudikan dengan perangkat bantu sebuah joystick kecil dan bukannya peralatan kendali konvensional. Berdasarkan alasan inilah A320 selalu mendapat perhatian khusus dalam masalah kesalahan yang diakibatkan oleh pengendalian komputer.

Pada saat sedang mendekati bandara Warsawa, sang pilot sedang mengharapkan suatu perubahan arah angin dan ini yang tak terjadi. Mereka bukannya mendarat dengan arah angin dari kepala (*headwind*) tetapi dari ekor (*tailwind*). Kemudian pilot mencoba menambah kecepatan dengan sengaja karena disebabkan badai. Hal ini mengakibatkan kecepatan terhadap tanah yang lebih tinggi. Pada jalur pacu (*runway*) pesawat ini memiliki gaya angkat (*lift*) yang terlalu besar. Sensor roda menginfromasikan pada komputer bahwa pesawat belum menyentuh tanah. Logika perancangan dari sistem pengereman menyimpulkan bahwa pesawat masih dalam kondisi terbang, dan tidak segera mengaktifkan mekanisme pengereman (*landing flap*, *reverse thrust,* dan *wheel brakes*). Airbus sial ini melewati landanan pacu dan menabrak sebuah bukit.

Paparan kebutuhan sistem (*requirement*) dan logika perancangan jelas menunjukkan ketidak- cocokkan. Paparan kebutuhan menyatakan 'ketika pesawat mencapai tanah, maka pesawat harus mampu mengerem', tetapi logika perancangan menyatkan 'hanya ketika sensor melaporkan nilai tertentu, maka rem dapat diaktifkan'.

Apakah ini sebuah kesalahan yang disebabkan komputer ? Jelas tidak merupakan kesalahan komputer sendiri saja, sebab cuaca, pengambilan keputusan dari awak pesawat, ramalan cuaca yang tak up-to-date, dan kondisi landansan pacu, bukit yang terletak pada ujung landasan pacu, juga merupakan faktor yang memberikan kontribusi kesalahan penyebab kecelakaan. Laporan resmi hanya memberikan penjelasan mengenai pengambilan keputusan awak berdasarkan cuaca, disain dari sistem pengereman, dan informasi yang kurang pada manual. Gosip tersebar bahwa telah terjadi diskusi hebat antara Lufthansa, Airbus, dan pejabat lalu lintas udara Jerman. Segera setelah itu perubahan dilakukan "mengikuti keinginan para pelanggan". Hal ini masuk akal karena, kita harus merubah apa yang dapat kita ubah, sebab perusahaan pembuat pesawat terbang hanya dapat melakukan perubahan amat sangat kecil terhadap cuaca dan landasan pacu.

Hubungan yang tepat antara realita ('pesawat telah mendarat') dan pengukuran yang dilakukan sensor (altitude, sensor tekanan pada badab pesawat) merupakan hal yang tak mudah untuk diterapkan. Mungkin perangkat kamera video dengan sistem pengenalan citra dapat menolong ? Tapi bagaimana bila dalam kondisi berkabut ? Dan apa yang terjadi bila modul ini tak berfungsi secara handal. Setiap kegagalan fungsi dari modul ini akan memberikan effek kepada modul lainnya , bahkan juga ketika modul-modul tersebut berfungsi dengan benar. Masalah ini tidak hanya pada Airbus saja, tetapi juga untuk pesawat-pesawat masa kini yang menggunakan sistem sensor tanah untuk pengeremannya.

**Kecelakaan Diakibatkan Kesalahan Kendali Komputer**

Pada bulan Juni 1994, sebuah pesawat A320 milik Dragonair di Hongkong menunjukkan kecelakaan yang benar-benar merupakan permasalahan komputer. Ketika mendekati bandara, wingflap harus mengubah posisinya, dari posisi sebelumnya. Pada saat yang sama, pesawat itu sedang terbang dan dilanda angin yang mencegah terjadinya perubahan posisi ini. Kemudian, wingflap berada pasa suatu posisi, dimana logika pengendali menganggap wingflap berada pada posisi sebaliknya. Pesawat mengalami kesulitan selama pendaratan. Berbeda dengan kasus Warsawa, masalah ini dimungkinkan timbul hanya pada kondisi ketika komputer mengendalikan pesawat. Malahan pada kasus lainnya tanpa kondisi tak berfungsinya sistem pada saat yang kritis, masalah komputer telah menyebabkan bencana kecelakaan. Kejadian ini menimpa pesawat Boeing B757 American Airlines yang terbang mendekati Cali, Columbia pada bulan Desember 1995. Pilot telah mengatur ketinggian menuju posisi lebih rendah, dan pada saat yang sama juga berusaha membenarkan rute dengan bantuan perangkat komputer manajemen penerbangan (*Flight Management Computer -- FMC*). Para awak pesawat mendapat gambaran yang salah mengenai posisi pesawat pada saat itu dan ditambah komunikasi dengan pengendali lalu lintas udara yang membingunkan. Para awak pesawat kurang berkonsentrasi pada penerbangan dan lebih berkonstrasi pada komputer sebab permasalahan disebabkan pada database navigasi. Digabung dengan fakta bahwa dua perangkat*beacon*pada saat yang bersamaan menggunakan identitas (ID) dan frekuensi yang sama. Akhrinya pesawat ini terbang menabrak gunung dan 160 nyawa menjadi korban sia-sia.

Berlawanan dengan makin kompleksnya pesawat terbang dan penanganannya, kecelakaan pesawat terbang modern "generasi ketiga" adalah lebih rendah dari sebelumnya, secara kasar dapat dikatakan satu kecelakaan per satu juta penerbangan. Walaupun demikian angka ini harus dikurangi lagi. Untuk itu dibutuhkan pemahanan yang lebih baik mengenai bagaimana komputer berinteraksi dengan komponen sistem yang lainnya, baik mekanik maupun dengan awak pesawat.(IMW)

**Oleh : Profesor Peter B Ladkin PhD**adalah expert bidang kecelakaan pesawat terbang dan pada saat ini mengepalai kelompok kerja penelitian di bidang jaringan dan sistem terdistribusi di Universitas Bielefeld - Jerman.

### Referensi

1. Ariane 5, Bericht der Untersuchungskomission,

<http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html>

1. WBA-Homepage, [http://www.rvs.uni-bielefeld.de](http://www.rvs.uni-bielefeld.de/)
2. Charles Perrow, Normal Accidents, Basic Books, 1984
3. Peter Ladkin (Hrsg.) Computer-related incident with commercial aircraft, <http://www.rvs.uni-bielefeld.de/~ladkin/Incidents/FBW.html>
4. Robyn. R. Lutz. Analyzing Software Requirements Errors in Safety-Critical Embedded Systems, IEEE International Symposium on Requirement Engineering, 1993, <http://www.cs.iastate.edu/~rlutz/homepage.html>
5. Pathfinder-Berichte, RISKS-Forum 19.49, 19.50, 19.53, 19.54, <http://catless.ncl.ac.uk/Risks/>

Computer Incident Advisory Capability, MIME Name Vulnerability in Outlook and Messenger, 1998, Bulleting I-077, [http://www.ciac.org](http://www.ciac.org/)