PENGUJIAN PERANGKAT LUNAK

-SOFTWARE DISASTER-

Eka Widhi Yunarso (EWD)
2017-2

Digunakan dilingkungan internal prodi D3 Manajemen Informatika, Fakultas Ilmu Terapan, Universitas Telkom

SOFTWARE DISASTER

1. ROKET MARINER I (1962)

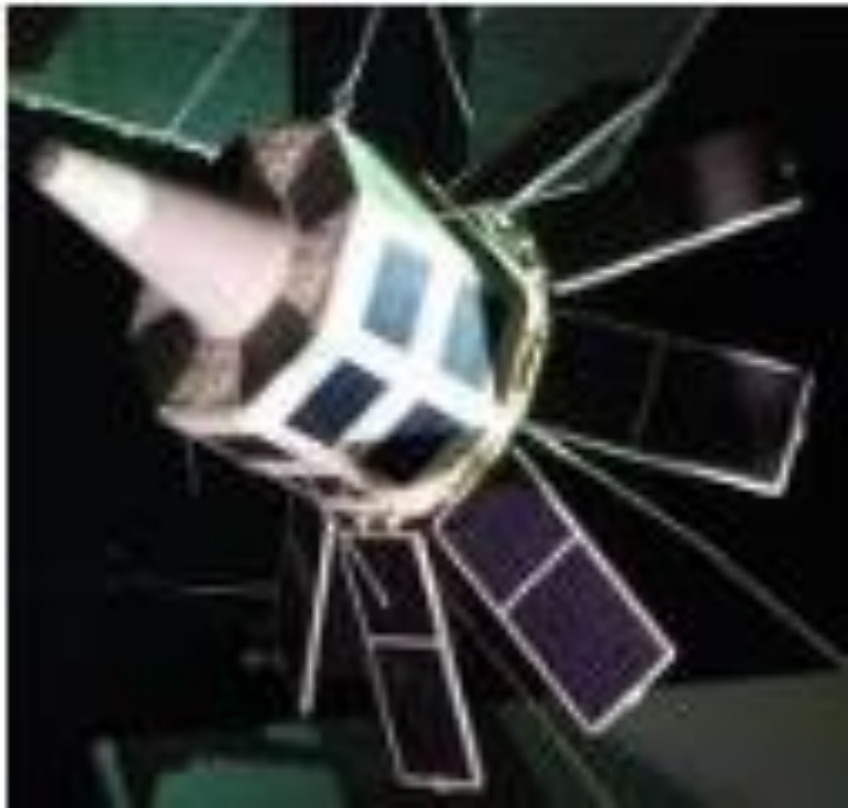
Roket ini keluar dari jalur peluncuran dan hancur.

Penyebabny: programmer lupa menuliskan kode overbar.

2. SATELIT EOLE I (1971)

Satelit milik Prancis ini jatuh dengan membawa 72 balon cuaca.

Penyebab: permintaan untuk mengirim data pengukuran diterjemahkan keliru oleh software sebagai perintah self-destroy.



3. SATELIT NIMBUS 7 (1978)

Satelit ini mengabaikan lubang ozon di atas Antartika.

Penyebab: software analisis menganggap nilai yang tidak lazim sebagai kesalahan dalam pengukuran dan kemudian mengoreksinya.

4. REAKTOR ATOM (1979)

Lima reaktor atom Amerika tidak berfungsi karena software memberikan nilai yang keliru untuk pengukuran kekuatan gempa bumi.

Penyebab: program melakukan penjumlahan yang seharusnya menghitung akar dari jumlah kuadrat.

5. GAS PIPELINE (1982 TOP SECRET)

Pipa gas di Siberia meledak akibat over-pressure.

Penyebabnya: Uni Sovyet membeli sebuah program kendali yang telah dimanipulasi oleh Amerika Serikat.

6. PERANG DUNIA KETIGA (1983)

Sebuah satelit Sovyet melaporkan adanya lima roket inter kontinental. Kolonel Petrow kemudian melihatnya sebagai ancaman walau ternyata keliru.

Penyebab: software mengartikan pantulan cahaya sebagai roket musuh.

7. THERAC 25 (1985-1987)

Alat rontgen menewaskan banyak pasien akibat dosis sinar yang berlebihan.

Penyebab: software hanya dapat melakukan beberapa tugas sekaligus dengan baik apabila pengguna memberikan perintah secara perlahan.

8. WALLSTREET (1987)

Bursa jatuh menimbulkan kerugian harian sebesar 22,6 % atau 500 milliar dollar.

Penyebab: software bursa tidak cukup cepat memproses order pembeli saham. Akibatnya terjadi kepanikan dalam transaksi.

9. USS VINCENNES (1988)

Kapal perang AS ini menembak sebuah pesawat penumpang Airbus milik Iran, 290 penumpang tewas.

Penyebab: sistem “Aegis” seharga 400 juta dollar melaporkan Airbus tersebut sebagai “assumed hostile”. Para crew kemudian menduga ada serangan pesawat tempur.

10. AT & T (1990)

Software telepon baru memaksa semua sentral swithching masuk ke dalam reset mode secara berantai. Oleh karena itu, telepon tidak dapat digunakan selama sembilan jam.

Penyebab: keliru menggunakan perintah “break”.

11. PATRIOT (1991)

Sistem senjata pertahanan ini gagal menghadang roket Scud. Akibatnya 28 tentara tewas.

Penyebab: kesalahan konversi dalam menghitung waktu. Nilai bertambah besar apabila sistem bekerja lebih lama.

12. DENVER AIRPORT (1995)

Sistem bagasi full-automatic baru ini tidak bekerja.

Penyebab: terlalu banyak perintah yang kompleks sehingga software overload.

13. ARIANE 5 (1996)

Roket ini keluar dari jalur dan meledak.

Penyebab: overflow saat mengonversi sebuah nilai floating 64 bit dalam software “Ariane 4”.

14. MARS CLIMATE ORBITER (1999)

Penjelajah ini terbakar di atsmofir Mars.

Penyebab: instrumen mengukur power satuan pound Inggris, sedangkan software NASA menggunakan satuan Newton.

15. MARS POLAR LANDER (1999)

Penjelajah ini jatuh ke Mars dengan kecepatan 80 km/jam dan hancur.

Penyebab: software mengartikan perintah untuk mengeluarkan kaki pendaratan sebagai status telah mendarat sehingga mematikan mesin.

16. BLACKOUT (2003)

Lantaran jaringan overload, listrik untuk 50 juta rumah tangga di AS dan Canada padam.

Penyebab: fungsi alarm dalam software menajemen listrik tidak bekerja.

17. HARTZ IV (2004)

Ratusan ribu penerima “Hartz IV” tidak memperoleh uang.

Penyebab: software mengisi nomor rekening penerima dari halaman yang salah sehingga diisi dengan nol.

18. AIRBUS A380 (2005)

Jet raksasa ini memakan biaya 5 milliar Euro lebih besar dan penyelesaiannya terlambat.

Penyebab: para designer pesawat ini menggunakan versi CAD-Software CATIA yang berbeda-beda satu sama lainnya.

19. BNP PARIBAS (2009)

Software milik Bank ini menjarah 10 ribuan rekening nasabahnya dengan melakukan 600.000 transaksi berkali-kali.

Penyebab: belum diketahui.

ARE THERE OTHERS DISASTERS?

Silahkan cari software disaster yang lain yang belum tercantum pada bahasan sebelumnya

TESTING IN SOFTWARE DEVELOPMENT LIFE CYCLE

PERJALANAN SOFT. ENGINEERING

Tahun 1940-an→Fokus utama: praktik dan teknologi untuk meningkatkan produktivitas para praktisi dan kualitas produk

Akhir 1950-an dan awal 1960-an baru mulai muncul istilah *Rekayasa Perangkat Lunak/Software Engineering* → masih ada perdebatan mengenai aspek engineering dari pengembangan perangkat lunak

PERJALANAN SOFT. ENGINEERING

Tahun 1968 dan 1969, komite sains NATO mensponsori dua konferensi tentang rekayasa perangkat lunak

Tahun 1960-an s.d 1980-an, banyak masalah yang ditemukan oleh para praktisi pengembangan perangkat lunak → proyek gagal → krisis perangkat lunak

Dimensi gagal = *duration time & budget*

PERJALANAN SOFT. ENGINEERING

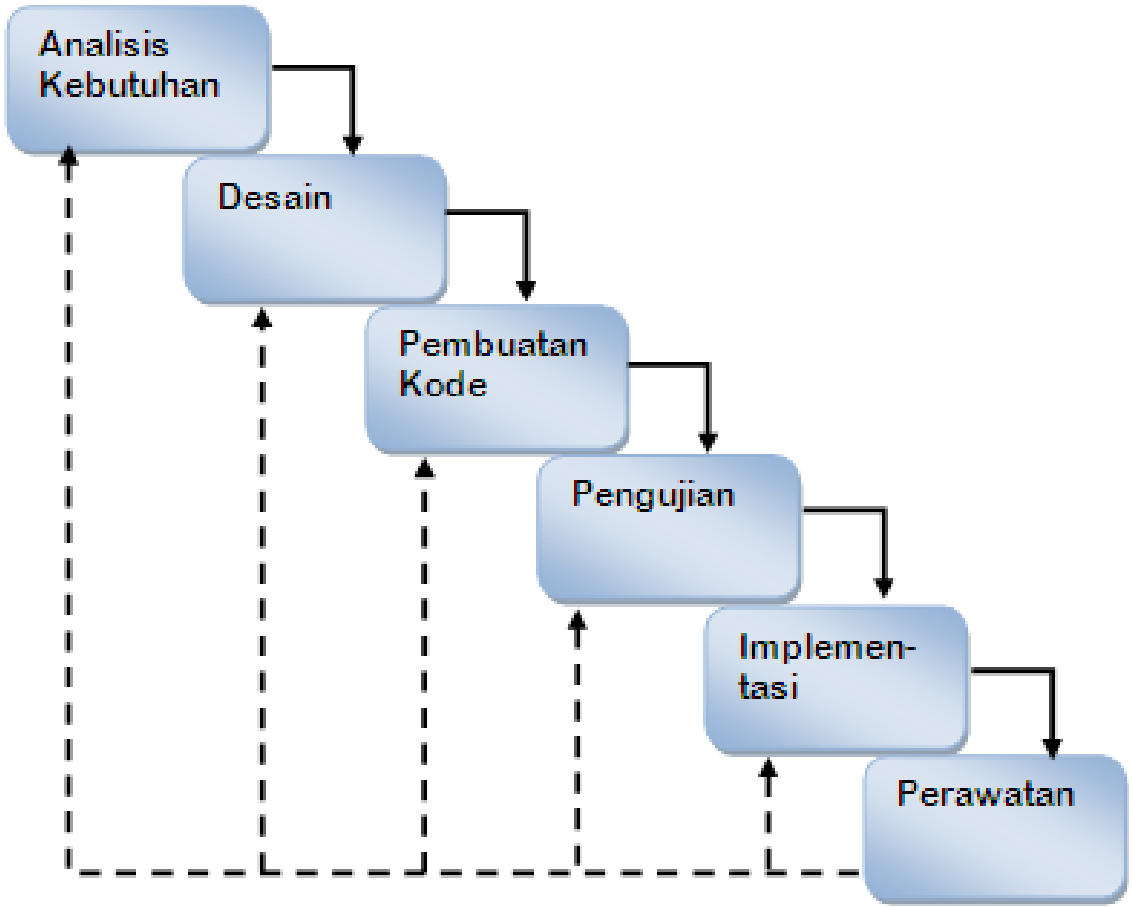
Selain kegagalan pada *time & budget* ada juga kegagalan yang berefek pada kerusakan fisik dan kematian. Contohnya: Roket Ariane 5 (Kegagalan terjadi saat dilakukan konversi dari bilangan *floating point* 64-bit ke bilangan *integer* 16-bit)

→ pentingnya ***TESTING***

SDLC PADA WATERFALL MODEL

Model klasik yang bersifat sistematis dan berurutan dalam membangun perangkat lunak → terilhami oleh proses manufaktur yang semua prosesnya sudah deterministik

SDLC PADA WATERFALL MODEL



Agar dapat dipergunakan terus menerus, *software* harus dipelihara dengan memperhatikan beberapa aspek:

- Mampu mengangani perkembangan data karena seiring berjalannya waktu.
- Mampu menangani ancaman kerusakan oleh virus atau program penyusup lainya.
- Mampu menangani perbaikan apabila ditemukan *error* atau *bug* pada aplikasi yang sedang dijalankan.
- Mampu menangani penambahan fitur baru.
- Mampu menangani perkembangan dan kemajuan teknologi.

QA PADA INFORMATION SYSTEM

Information System pada pembahasan adalah C*omputer Based Information System* (CBIS). Pada CBIS *Software* menjadi bagian yang penting dan menjadi ciri khas dari CBIS itu sendiri.

Software Quality assurance berada pada pada semua tahapan SDLC

Cari definisi *Software Quality Assurance*

QA PADA INFORMATION SYSTEM

Pada Tahap Analisis Kebutuhan

Apakah *Client* sudah memahami dengan baik apa yang dia butuhkan.

Apakah desain yang dibuat oleh *System Analist* sudah mengkover semua yang diinginkan?

QA PADA INFORMATION SYSTEM

Pada Tahap Desain

Apakah Desainer sudah benar menggambarakan seluruh kebutuhan *Client*?

Apakah Desain yang dibuat cukup jelas dan detil untuk diimplementasikan dalam *source code* dan basis data secara tepat tanpa multi tafsir?

Contoh: desain halaman login

QA PADA INFORMATION SYSTEM

Pada Tahap Pembuatan Kode Program

Apakah *programmer* sudah mengimplemtnasikan desain secara benar?

Apakah implementasi yang digunakan sudah menggunkan *tool* yang optimal?

Apakah dalam implementasi desain ke sumber kode program, *programmer* sudah mengindahkan kaidah-kaidah yang benar?

QA PADA INFORMATION SYSTEM

Pada pembuatan kode sumber program ada beberapa hal yang dapat dievaluasi untuk menentukan kualitasnya:

- *Readable*
- *Bugfree*
- *Modular*
- Sesuai dengan waktu dan *budget*
- Dapat dikembangkan lebih lanjut
- Dapat di pelihar perkembangannya
- Diimplementasikan berdasarkan desain yang jelas

QA PADA INFORMATION SYSTEM

Pada Tahap Pengujian Program

Pengujian dilakukan untuk mendapatkan *error* dari program yang sudah dibuat.

Pada pengujian terdapat beberapa kaidah untuk mendapatkan pengujian yang baik [Roger Pressman]

QA PADA INFORMATION SYSTEM

Pada Tahap Implementasi

Seringkali komputer tempat pembuatan aplikasi (sering disebut sebaga komputer *development* / devel) berbeda lingkungannya dengan komputer implementasi. Perbedaan *hardware* maupun *software* akan berpengaruh kepada jalannya aplikasi yang dijalankannya.

QA PADA INFORMATION SYSTEM

Pada Tahap Perawatan

Untuk mempertahan kinerja perangkat lunak diperlukan perawatan terhadap perangka lunak tersebut. Dengan pemeliharaan, perangkat lunak dapat dipertahankan penggunaanya, dan dipertahakankan atau bahkan ditingkatkan performansinya.

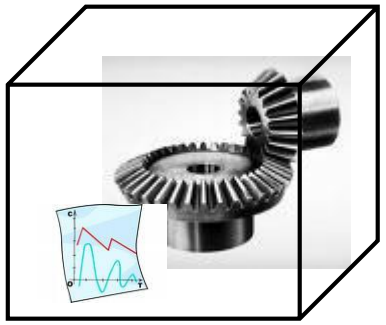
JENIS-JENIS PENGUJIAN PL

Black box testing adalah pengujian yang dilakukan dengan cara mengamati hasil eksekusi melalui data uji dan memeriksa fungsional dari perangkat lunak.

mengevaluasi perangkat lunak hanya dari tampilan luarnya saja (interface), fungsionalitasnya, tanpa mengetahui apa yang sesungguhnya terjadi dalam proses detil yang ada dibalik itu.

JENIS-JENIS PENGUJIAN PL

Pada whitebox *testing*, pengujian dilakukan sampai pada level detil dari suatu perangkat lunak, yaitu *source code*.



Sebuah kota putih yang jernih. Anda tahu apa saja yang ada di dalamnya dan melihat dengan jelas bagaimana pengolahan masukan diproses hingga menghasilkan keluaran.

JENIS-JENIS PENGUJIAN PL

Factory *Acceptance Test* vs. *User Acceptance Test*

Uji terima perangkat lunak yang dilakukan di tempat pengembangan perangkat lunak disebut Factory Acceptance Test (FAT).

Uji terima perangkat lunak yang dilakukan di tempat pengguna (User) perangkat lunak disebut User Acceptance Test (UAT).

JENIS-JENIS PENGUJIAN PL

Alpha *Test* Vs. Betha *Test*

Pengujian terhadap perangkat lunak yang siap untuk dipasarkan dibawah kendali programmer maupun developer disebut Apha Test. Perangkat lunak yang sedang diuji pada tahap ini sering disebut sebagai Release Alpha

Pengujian terhadap perangkat lunak yang siap untuk dipasarkan dan dilakukan oleh sebagian user di pasar tersebut tanpa pengawasan developer disebut Betha Test. Perangkat lunak yang sedang diuji pada tahap ini sering disebut sebagai Release Betha

JENIS-JENIS PENGUJIAN PL

Stress *Test*

Pengujian uni dilakukan dengan memberi beban pada perangkat lunak untuk dapat diketahui titik maksimum performansi perangkat lunak.

Stress test sering dilakukan pada aplikasi yang membutuhkan konkurensi maupun akses acak yang bersamaan dalam jumlah yang sangat banyak.

Aplikasi dengan berbasis web dengan *request* yang sangat banyak menjadi contoh yang baik dalam hal ini