ESTETIKA DAN KUALITAS PERANGKAT LUNAK

Oleh : Kusrini, S.Kom.

Abstract

Building a software system is not only talking about algorithm but also art aspect. An aesthetic valued softwares will have great advantages : working better, cost cheaper, match user requirement, have fewer bugs, run faster, easier to fix, and have longer life cycle. There are some principle for aesthetic software that is discussed in this paper : cooperation, appropriate form, system minimality, component singularity, functional locality, readibility and simplicity.

Keywords : aesthetic, software, art, standard, development, design

Pendahuluan

Ada banyak perangkat lunak yang dirancang seadanya. Apabila desain perangkat lunak itu dimisalkan sebuah jembatan, maka tidak ada seorang yang berakal sehat pun yang akan menyeberanginya. Apabila diumpamakan sebuah rumah, maka kita akan takut untuk memasukinya. Satu-satunya alasan mengapa seorang perangkat lunak engineer bisa bertahan dengan keadaan ini adalah karena publik tidak bisa melihat dengan mudah ke dalam sistem suatu perangkat lunak. Jika suatu desain perangkat lunak bisa dilihat seperti jembatan atau rumah, mungkin kita akan bersembunyi karena malu.

Kita tidak akan menerima suatu rumah baru yang lantainya kotor, lubang di langit-langit, paku berantakan di tembok, --- dan harga yang mahal. Walaupun rumah tersebut memenuhi kebutuhan minimal kita. Kita tidak akan tertarik dengan penjelasan : “Yah, rumahnya ada pintu depannya, yang biasanya selalu terbuka. Anda dapat berjalan ke dapur, tapi hati-hati dengan paku-paku di tembok. Lubang di langit-langit tidak seberapa bocor kok. Dan memang sih pembangunannya overbudget 300%, tapi kan memang biasanya rumah seperti itu…”. Pada perangkat lunak, manifestasi dari desain yang buruk adalah redundansi, *bottleneck* kinerja yang tidak perlu, *bug-bug* saling berkaitan yang tidak bisa diperbaiki, kode yang tidak bisa dipahami, dan penyakit-penyakit lainnya. Sayangnya, kita biasanya menerima saja perangkat lunak dengan kondisi seperti itu. Biasanya, perusahan-perusahaan perangkat lunak merilis kode seperti ini ke pelanggan eksternal dan internal. Dan pelanggan menerimanya, kemudian bisnis-bisnis membayar jutaan rupiah per tahunnya untuk perangkat lunak semacam ini.

Artikel ini menantang para *engineer*, manajer, eksekutif dan pengguna perangkat lunak (semua orang) untuk meningkatkan standar kita tentang perangkat lunak. Kita seyogyanya mengharapkan level kualitas dan kinerja yang sama dalam perangkat lunak seperti yang kita minta pada konstruksi fisik. Alih-alih mencoba membuat perangkat lunak yang bekerja minimalis, kita justru mestinya membuat perangkat lunak yang memiliki kecantikan internal. Program yang cantik bekerja lebih baik, biaya lebih murah, cocok dengan kebutuhan user, memiliki *bug* yang lebih sedikit, berjalan lebih cepat, lebih mudah diperbaiki dan memiliki masa hidup lebih lama. Meningkatkan standar kita tentang perangkat lunak adalah bukan suatu hal berlebihan yang hanya layak untuk programer yang memiliki waktu luang. Membuat perangkat lunak menyenangkan dan estetis adalah hal yang krusial untuk membuat perangkat lunak yang lebih baik dan lebih murah.

Menurut Connel, estetika merupakan suatu penilaian yang bersifat kualitatif, namun, seperti juga arsitektur fisik, estetika perangkat lunak terdiri dari beberapa prinsip umum, yaitu :

- Kooperatif
- Bentuk yang Sesuai

- Minimalitas Sistem
- Ketunggalan Komponen
- Lokalitas Fungsional
- *Readability*
- Simplisitas

Kooperatif

Seperti bangunan yang harus didesain untuk menyatu dan memperindah lingkungannya, perangkat lunak juga mesti bisa bekerja sama secara baik dengan lingkungannya. Jika dimisalkan lingkungan untuk sebuah bangunan, adalah kemiringan lahan, orientasi terhadap matahari, cuaca lokalnya, bangunan lain yang berdekatan dan lain sebagainya. “Lingkungan” dari suatu sistem perangkat lunak adalah hardware komputer, sistem operasi, middleware lainnya (seperti database atau sistem keamanan), aplikasi perangkat lunak lainnya pada komputer, dan gaya berinteraksi dari user pengguna.

Contoh dari sistem perangkat lunak yang kooperatif dengan lingkungannya :

- Suatu buku saku appointment elektronik yang bekerja dalam batasan layar kecil hitam putihnya, dan masih dapat menampilkan informasi dengan jelas.
- Suatu sistem pembukuan pajak penghasilan yang bisa dibayar lewat Internet, karena banyak sekali user yang sudah memiliki akses Internet.
- Suatu tutorial membaca untuk siswa tingkat dua yang bisa berjalan dengan cepat pada komputer tua, karena banyak sekolahan yang masih hanya memiliki komputer seperti itu.
- Sebuah sistem manajemen dokumen berskala besar. Arsitek perangkat lunaknya dengan cerdas mendesain metode penyimpanan dan pengambilan data yang mendukung pengoperasian disk drive komputer, sehingga keseluruhan sistem dapat bekerja seefisien mungkin.

Dari semua kasus-kasus di atas, perangkat lunak-perangkat lunak tersebut kooperatif dengan lingkungannya – seperti suatu bangunan yang didesain dengan baik, berjalan harmonis dengan lingkungannya

Bentuk yang Sesuai

Desain internal dari suatu sistem perangkat lunak harus merefleksikan dan meng-create fungsi eksternalnya. Suatu bangunan yang indah menyatukan bentuk dan fungsi, begitu pula dengan perangkat lunak yang baik. Mengapa ini menjadi masalah? Selagi suatu sistem perangkat lunak bekerja dengan benar, apakah hasil dari suatu internal desain menjadi masalah? Jawabannya adalah Ya. Sebuah struktur internal yang mempertimbangkan fitur eksternal akan cenderung meng-create fitur tersebut lebih baik. Sebuah bentuk perangkat lunak yang mengikuti fungsi perangkat lunak tersebut, juga cenderung lebih simpel, karena fitur eksternal tersebut diangkat dari desain internal daripada dipaksakan diatas desain tersebut. Suatu sistem perangkat lunak yang bentuknya tidak mencerminkan fungsinya selamanya akan sulit untuk di debug, memiliki lebih banyak bug, dan sulit untuk dikembangkan dan dimodifikasi, serta cenderung berkinerja buruk pada fungsi-fungsi intinya.

Karena kita tidak dapat menyentuh perangkat lunak, terkadang sulit untuk menilai apakah suatu bentuk sistem perangkat lunak cocok dengan fungsinya. Namun seluruh perangkat lunak memiliki suatu bentuk

yang pasti. Ambil contoh suatu sistem akuntansi. Akuntansi bisnis terdiri dari beberapa operasi-operasi yang telah didefinisikan: pembelian, billing, payroll, general ledger, dan lain-lain. Untuk suatu penggunaan luas, fungsi-fungsi ini terpisah, namun ada pula overlap diantaranya. Suatu sistem perangkat lunak untuk akuntansi bisnis telah merefleksikan operasi-operasi akunting logis ini dalam desain internalnya. Harus ada suatu bagian terdefinisi yang jelas dari perangkat lunak tersebut untuk pembelian, billing, payroll, general ledger, dan lain-lain. Harus ada pula suatu overlap yang jelas dalam perangkat lunak tersebut dimana overlap operasi logis tersebut. Tanpa desain perangkat lunak seperti itu, sangatlah tidak mungkin untuk mengubah hanya satu aspek dari payroll tanpa mempengaruhi aspek lainnya, dalam operasi yang tidak berhubungan. Bentuk yang sesuai juga membuat engineer dapat mengubah dalam suatu area yang meng-overlap area lainnya (seperti general-ledger) dan mengubah semua operasi yang berhubungan dengan benar.

Perancangan perangkat lunak yang menggambarkan fungsi eksternal memberikan arti bahwa tidak ada aturan yang baku untuk teknik pemrograman yang baik. Selama bertahun-tahun, programmer diajari bahwa variabel global dan statemen GOTO merupakan praktek pemrograman yang buruk. Kenyataannya, dalam beberapa kondisi, struktur ini bisa jadi merupakan hal yang persis dibutuhkan perangkat lunak untuk mengawinkan bentuk dengan fungsi. Adalah suatu hal yang salah apabila mengikuti aturan pembangunan : “SELALU GUNAKAN KAYU JATI DARIPADA KAYU PINUS”. Jati adalah kayu yang sangat bagus, namun terkadang kayu pinus adalah pilihan yang benar. Dengan cara yang sama, pertanyaan yang benar untuk teknik pemrograman adalah : “Apakah ini adalah suatu rancangan yang benar untuk fitur-fitur eksternal yang kita inginkan?”

Minimalitas Sistem

Jika dibayangkan sebuah rumah yang sedang dibangun di jalan yang berfasilitas saluran air bersih dan listrik. Sekarang umpama pembangun rumah menggali sebuah sumur pribadi atau membuat pembangkit listrik sendiri. Ketika ditanya mengapa mereka melakukan ini, jawabannya adalah : “Kami ingin sumur kami sendiri, Sehingga kami bisa menggunakannya sesuka hati. Kemudian kami sendiri tidak tahu bahwa ada jaringan listrik di jalan ini”. Pekerjaan dan pengeluaran yang berlebihan ini tidak bisa diterima. Memang tidak ada salahnya menambahkan sumur atau generator listrik di suatu rumah – apabila rumah tersebut benar-benar membutuhkannya. Adalah suatu hal yang mengerikan apabila menyertakan hal-hal tersebut untuk alasan kecil atau tidak perduli akan adanya kemungkinan fasilitas-fasilitas umum yang tersedia. Rancangan bangunan yang baik berusaha menjaga bangunan tersebut seminimal mungkin, dengan menggunakan sumber daya eksternal yang tersedia.

Perangkat lunak yang cantik juga mengikuti prinsip yang sama – sekecil mungkin, dengan menggunakan sumber daya komputasi yang tersedia apabila dimungkinkan. Adalah tanggungjawab dari setiap perancang perangkat lunak dan engineer untuk mengerti sistem komputasi yang mereka gunakan, dan menggunakan fasilitasnya apabila tersedia. Perangkat lunak haruslah hanya berisi apa yang dibutuhkan saja, tidak lebih.

Sebagai sebuah contoh kasus adalah sebuah proyek sistem informasi keuangan yang akan dijalankan pada minikomputer VAX. Pengembang programnya memimilih untuk membangun semuanya dari awal, daripada harus menggunakan fasilitas yang sudah terkandung di dalam komputer VAX tersebut. Mereka menulis prosedur-prosedur pengurutannya sendiri, paket input/output layar, kontrol source-code dan tool-tool otomatis – semua yang mana hal-hal tersebut padahal disediakan oleh VAX. Kejadian ini ternyata bermula dari kemalasan dan ketidakpedulian pengembang perangkat lunak terhadap komputer yang mereka gunakan. Sistem hasilnya menjadi berkali lipat lebih besar dari seharusnya, berjalan dengan lambat, jadi dalam waktu yang lebih lama dan membutuhkan biaya yang jauh lebih besar.

Ketunggalan Komponen

Umumnya, bangunan yang cantik berisi ruangan-ruangan yang setiap ruangan tersebut berguna untuk memenuhi satu tujuan tertentu dengan baik. Sebagai contoh, hampir setiap rumah memiliki sebuah kamar tidur utama, yang berisi apa-apa yang dibutuhkan oleh ruangan tersebut. Adalah suatu rancangan yang buruk bila sebuah rumah sampai memiliki empat kamar tidur utama karena pembangunnya tidak dapat menyelesaikan satupun ruangan. Lebih buruk lagi, apabila pembangun lupa bahwa mereka baru saja membuat sebuah kamar tidur utama, lalu mereka membangun lagi satu lagi, kemudian mereka lupa lagi, dan membangun lagi dan seterusnya.

Tentu saja, beberapa bangunan-bangunan yang besar mungkin membutuhkan lebih dari satu ruangan yang berfungsi sama. Sebagai contoh, suatu bangunan kantor dengan 5000 pekerja bisa jadi membutuhkan lebih dari satu kafetaria. Namun hal ini telah dipertimbangkan masak-masak karena memang dibutuhkan.

Secara analogis, perangkat lunak yang dirancang baik umumnya berisi satu instan untuk masing-masing komponen, dan membuat komponen tersebut berjalan dengan benar. Lawannya adalah redundansi dan dikenal dengan arsitektur perangkat lunak yang buruk. Sebagai contoh adalah sebuah sistem perangkat lunak yang memiliki tiga driver untuk satu printer yang sama. Setiap perubahan pada printer membutuhkan perubahan untuk 3 bagian dari perangkat lunak. Banyak sekali, ternyata, sistem-sistem perangkat lunak yang mengandung kode-kode redundan dan tidak perlu. Dalam banyak kasus hal ini disebabkan karena seorang programer tidak tahu bahwa engineer yang lain telah menyelesaikan masalah yang sama. (Pembangun-pembangun lupa bahwa mereka telah membuat kamar tidur utama).

Lokalitas Fungsional

Rancangan pembangunan yang baik menempatkan item-item yang berhubungan pada tempat yang berdekatan. Peralatan dan bahan-bahan untuk menyiapkan makanan biasanya terletak dalam ruang yang sama. Peralatan-peralatan rias, silet cukur, deodoran biasanya terletak berdekatan di meja rias. Walaupun suatu rumah dapat dibangun dengan kulkas di loteng, atau oven di ruang keluarga, dan mesin cuci di dalam kamar tidur – namun ini termasuk rancangan yang buruk.

Rancangan perangkat lunak yang baik mengikuti prinsip yang sama dalam menempatkan hal-hal yang berhubungan di tempat yang berdekatan. Ketika perangkat lunak dibangun seperti ini, maka sangat mudah untuk memahami perangkat lunak bagi tim proyek – karena strukturnya bisa dinalar dengan mudah pula. Untuk memperbaiki bug dan membuat perubahan sangat mudah karena kode-kode yang relevan terletak di tempat yang terlihat.

Lokalitas Fungsional berkaitan dengan level abstraksi. Setiap ruang dalam suatu rumah memiliki tujuan dengan hal-hal yang berkaitan untuk tujuan tersebut terletak dalam ruang itu. Dalam pandang pemikiran yang lebih luas, arsitektur bangunan yang baik menempatkan ruang-ruang yang berkaitan pada lokasi yang berdekatan. Ruang-ruang untuk aktifitas sehari-hari terletak pada bagian tertentu (atau dalam satu lantai). Ruang-ruang yang digunakan pada malam hari terletak pada bagian yang lain atau di lantai atas. Bangunan-bangunan kantor menempatkan ruang infrastruktur mekanik dalam satu area, jauh dari tempat bisnis. Arsitektur perangkat lunak yang baik juga menggunakan level-level abstraksi untuk mendapatkan Lokalitas Fungsional. Dalam sebuah sistem operasi, seluruh kode level rendah untuk efek audio (suara) harus terletak dalam satu kelompok tempat. Di dalam setiap level fitur yang lebih tinggi (seperti file-system), kode untuk efek-efek suara tersebut juga harus dikelompokkan. Berlanjut ke level yang lebih tinggi, kode-

kode untuk fitur-fitur yang berkaitan (seperti file-system dan Internet Explorer dalam Windows) harus dikelompokkan pula.

Readability

Dua program perangkat lunak bisa jadi memiliki fungsi yang sama, dan memiliki rancangan internal dan konstruksi yang sama pula dari suatu perspektif teknis, namun berbeda jauh dari sisi *readibility* manusia.

Ada dua aspek pada *readibility* perangkat lunak: yang pertama yaitu kejelasan yang dibangun ke dalam kode, dan yang kedua adalah comment yang memberi catatan kode tersebut. Hal yang pertama termasuk penamaan variabel dan konstanta yang memberikan arti, penggunaan yang tepat untuk jeda spasi dan indenting, struktur kontrol yang transparan, dan path-path eksekusi normal yang straight-line. Praktek pemberian comment yang baik menekankan pada comment yang dapat memberikan pemahaman pada programer selanjutnya tentang topik-topik yang tidak dapat diambil dari kode itu sendiri, seperti maksud dari setiap modul.

Readibility adalah satu area dalam estetika perangkat lunak yang tidak memiliki kesejajaran dengan konstruksi fisik, tapi adalah sangat penting. Apabila pengembang-pengembang perangkat lunak tidak dapat memahami source code dari engineer, maka kode tersebut secara efektif tidak mengandung metrik-metrik lain yang didiskusikan di sini. Kualitas-kualitas tersebut mungkin ada pada beberapa pengertian teknis, namun ketidakmungkinan suatu kode untuk ditelusuri ke dalamnya membuat kode tersebut tidak dapat digunakan untuk tujuan-tujuan praktis. Sebagai contoh, diambil suatu himpunan file-file source menunjukkan minimality yang sempurna (tidak mengandung redundansi). Namun apabila tidak ada manusia yang dapat menemukan bagian tertentu dari kode sumber yang berhubungan dengan fitur tertentu, maka tidak ada seorangpun yang dapat memperbaiki bug-bug di lokasi tersebut atau mengembangkan fitur-fitur dengan jalan apapun.

Simplisitas

Perangkat lunak seharusnya mengerjakan pekerjaannya dan memecahkan masalah dengan cara sesimpel mungkin. Dalam banyak cara, simplisitas adalah prinsip yang paling penting dari semuanya dan melingkupi prinsip-prinsip lainnya. Perangkat lunak yang simpel adalah hal yang cantik. Perangkat lunak yang cantik berbentuk simpel. Program-program yang simpel memiliki bug yang lebih sedikit (karena baris-baris kode yang mungkin salah lebih sedikit pula), berjalan lebih cepat (karena hanya mengandung instruksi mesin yang lebih sedikit), lebih kecil (karena kode yang terkompilasi lebih sedikit), dan lebih mudah untuk diperbaiki apabila mengandung bug (karena lebih sedikit tempat yang harus diperiksa untuk diperbaiki). Oleh karena itu, program-program yang sederhana jauh lebih murah untuk dibuat dan dirawat.

Simpicity dari perangkat lunak juga merupakan suatu metrik kunci yang membedakan kemampuan programing. Programer junior menciptakan solusi simpel untuk masalah simpel. Programer senior menciptakan solusi kompleks untuk masalah kompleks. Programer besar dapat menemukan solusi simpel untuk masalah yang kompleks. Kode yang ditulis oleh seorang programer top akan terlihat jelas, apabila sudah selesai, namun lebih sulit untuk dibuat. Sejalan dengan tujuan ilmu pengetahuan yaitu menyederhanakan dan mengurutkan hal-hal dalam alam semesta yang kompleks, maka tujuan dari pemrograman haruslah untuk menemukan solusi sederhana dari suatu masalah yang kompleks.

Dr. Indrajit juga menyampaikan factor-faktor yang memperngaruhi kualitas perangkat lunak usulan dari McCall dan kawan-kawan (1997). Faktor-faktor tersebut adalah:

1. Sifat-sifat operasional dari software (Product Operations);

2. Kemampuan software dalam menjalani perubahan (Product Revision); dan
3. Daya adaptasi atau penyesuaian software terhadap lingkungan baru (Product Transition).

Product Operations

Sifat-sifat operasional suatu software berkaitan dengan hal-hal yang harus diperhatikan oleh para perancang dan pengembang yang secara teknis melakukan penciptaan sebuah aplikasi. Hal-hal yang diukur di sini adalah yang berhubungan dengan teknis analisa, perancangan, dan konstruksi sebuah software. Faktor-faktor McCall yang berkaitan dengan sifat-sifat operasional software adalah:

• Correctness – sejauh mana suatu software memenuhi spesifikasi dan mission objective dari users;

• Reliability – sejauh mana suatu software dapat diharapkan untuk melaksanakan fungsinya dengan ketelitian yang diperlukan;

• Efficiency – banyaknya sumber daya komputasi dan kode program yang dibutuhkan suatu software untuk melakukan fungsinya;

• Integrity – sejauh mana akses ke software dan data oleh pihak yang tidak berhak dapat dikendalikan; dan

• Usability – usaha yang diperlukan untuk mempelajari, mengoperasikan, menyiapkan input, dan mengartikan output dari software.

Product Revision

Setelah sebuah software berhasil dikembangkan dan diimplementasikan, akan terdapat berbagai hal yang perlu diperbaiki berdasarkan hasil uji coba maupun evaluasi. Sebuah software yang dirancang dan dikembangkan dengan baik, akan dengan mudah dapat direvisi jika diperlukan. Seberapa jauh software tersebut dapat diperbaiki merupakan faktor lain yang harus diperhatikan.

Faktor-faktor McCall yang berkaitan dengan kemampuan software untuk menjalani perubahan adalah:

• Maintainability – usaha yang diperlukan untuk menemukan dan memperbaiki kesalahan (error) dalam software;

• Flexibility – usaha yang diperlukan untuk melakukan modifikasi terhadap software yang operasional

• Testability – usaha yang diperlukan untuk menguji suatu software untuk memastikan apakah melakukan fungsi yang dikehendaki atau tidak

Product Transition

Setelah integritas software secara teknis diukur dengan menggunakan faktor product operational dan secara implementasi telah disesuaikan dengan faktor product revision, faktor terakhir yang harus diperhatikan adalah faktor transisi – yaitu bagaimana software tersebut dapat dijalankan pada beberapa platform atau kerangka sistem yang beragam.

Faktor-faktor McCall yang berkaitan dengan tingkat adaptibilitas software terhadap lingkungan baru:

- Portability – usaha yang diperlukan untuk mentransfer software dari suatu hardware dan/atau sistem software tertentu agar dapat berfungsi pada hardware dan/atau sistem software lainnya.
- Reusability – sejauh mana suatu software (atau bagian software) dapat digunakan ulang pada aplikasi lainnya
- Interoperability – usaha yang diperlukan untuk menghubungkan satu software dengan lainnya

Kesimpulan

Akhirnya, kualitas global dari perangkat lunak yang cantik yang bukanlah jumlahan begitu saja dari atribut-atribut sebelumnya. Untuk membuat sebuah perangkat lunak yang berkualitas harus diperhatikan kecantikan desain perangkat lunak tersebut. Dengan desain yang memadai diharapkan program akan bekerja lebih baik, memerlukan biaya lebih murah, cocok dengan kebutuhan user, memiliki *bug* yang lebih sedikit, berjalan lebih cepat, lebih mudah diperbaiki dan memiliki masa hidup lebih lama.

Daftar Pustaka

Connell, C. 1998, *Most Software Stink*, http://www.chc-3.com/pub/beautifulsoftware.htm. diakses tanggal 10 Agustus 2005

Indrajit, R.E., Dr. Ir.M.Sc. M.B.A, 2004, Kiat Memilih Software, eBizzAsia, http://www.ebizzasia.com/0218-2004/q&a,0218.html, diakses tanggal 15 Agustus 2005