Paradigma,Proses Desain Dan Rekayasa Daya Guna

PARADIGMA DALAM IMK

Pendahuluan

Tujuan utama dari suatu sistem interaktif adalah memungkinkan user mencapai suatu tujuan tertentu dalam suatu domain aplikasi. Sehingga untuk mencapai tujuan tersebut, sebuah sistem interaktif harus dapat didayagunakan .

Berdasarkan hal tersebut, muncul dua pertanyaan bagi perancang sistem interaktif, yaitu :

1) Bagaimana suatu sistem interaktif dibuat / dibangun supaya mempunyai dayaguna yang tinggi ?
2) Bagaimana mengukur atau mendemonstrasikan dayaguna suatu sistem interaktif ?

Kedua pertanyaan tersebut dapat dijawab dengan digunakan dua pendekatan, yaitu :

1) Menggunakan contoh dari sistem-sistem interaktif yang telah dibangun sebelumnya dan diyakini berhasil / sukses dalam meningkatkan dayaguna sistem tersebut. Hal ini disebut sebagai **paradigma interaksi** untuk pengembangan sistem interaktif di masa depan.

2) Menggunakan berbagi **prinsip interaksi** efektif dari berbagai aspek pengetahuan psikologi, komputasi dan sosiologi mengarahkan peningkatan desain dan evolusi suatu produk, yang pada akhirnya akan meningkatkan dayaguna sistem tersebut.

Perbedaan yang terjadi antara ***Paradigma*** dan ***Prinsip*** merupakan suatu refleksi yang penting dalam sejarah bidang Interaksi Manusia dan Komputer (IMK). Refleksi ini dipakai untuk membangun sistem yang lebih baik di masa depan.

Paradigma interaksi yang ada, sebagian besar memanfaatkan kemajuan teknologi komputer dan membangun aplikasi yang kreatif untuk meningkatkan kualitas interaksi.

Sedangkan ***Prinsip interaksi*** terlepas dari kemajuan teknologi, dan lebih menggali pada pemahaman aspek manusia pada proses interaksi.

Paradigma Interaksi

Kemajuan dalam bidang IMK diperoleh dari usaha eksplorasi dan kreatifitas rancangan yang dibuat. Pada bagian ini akan **dibahas kelebihan-kelebihan** dari sisi tehnik dan rancangan pada beberapa sistem interaksi yang dianggap sebagai kemajuan dalam bidang IMK.

1) Time-Sharing

Pada tahun 1940 dan 1950-an, terjadi perkembangan yang siginifikan dalam teknologi perangkat keras komputer. Hingga pada tahun 1960-an, perkembangan teknologi hardware yang cepat ini kelihatan menjadi sia-sia jika tidak diimbangi dengan pemanfaatannya, dan mendorong para peneliti untuk mencari ide-ide baru yang akan diaplikasikan pada perkembangan teknologi perangkat keras komputer yang cepat tersebut.

Salah satu kontribusi yang besar pada masa itu adalah konsep *time-sharing* yang memungkinkan sebuah komputer mampu mendukung / dapat digunakan oleh banyak (*multiple*) user.

Sebelumnya, user / programmer dibatasi oleh pemrosesan batch, dengan memberikan data atau instruksi yang akan dijalankan dalam bentuk *punched card* atau *paper tape* kepada operator yang akan memasukkannya ke dalam komputer.

2) **Video Display Units (VDU)**

Pada pertengahan tahun 1950-an, para peneliti bereksperimen untuk dapat menampilkan /mempresentasikan dan memanipulasi informasi pada komputer dalam bentuk citra (*image*) pada video display unit (VDU).

Tampilan pada layar merupakan media yang lebih baik daripada cetakan pada kertas untuk menyajikan informasi strategis dalam jumlah besar yang digunakan pada pemrosesan cepat.

Hingga pada tahun 1962, Ivan Sutherland menciptakan sebuah software "*Sketchpad*" yang dapat digunakan lebih dari sekedar pemrosesan data. Software ini memungkinkan user melakukan abstraksi data dalam beberapa tingkat detail, memvisualisasikan dan memanipulasi representasi yang berbeda dari informasi yang ada.

3) **Programming Toolkits (Alat Bantu Pemrograman)**

Sekitar awal tahun 1950-an, komputer dianggap sebagai suatu teknologi yang kompleks sehingga hanya orang dengan intelektualitas tertentu saja yang mampu memanipulasinya.

Oleh karena itu, peralatan komputasi untuk membantu manusia dalam memecahkan masalah perlu dilengkapi dengan alat bantu (*tools*) yang tepat. Untuk itu, diadakan riset dengan sebuah tim untuk membangun alat bantu pemrograman (*programming tools*). Dari alat bantu

Pemrograman ini dapat dibuat alat bantu lain yang lebih besar cakupannya dan akhirnya programer dapat membangun sistem interaktif atau sistem lain yang lebih kompleks.

4) Komputer Pribadi (Personal Computing)

Engelbart mempunyai visi bahwa komputer tidak hanya diperuntukkan bagi mereka yang mengerti komputer saja. Salah satu hasil awalnya adalah software LOGO yang dibuat oleh Seymour Papert. Software ini mengajarkan anak-anak bahasa pemrograman grafis yang mudah dengan menggunakan analogi cursor dalam bentuk ekor kura-kura dan frasa bahasa Inggris.

Dengan mengadaptasi bahasa pemrograman grafis yang dapat dimengerti dan digunakan oleh anak-anak, menunjukkan bahwa nilai utama dari sebuah interaksi tidak terletak pada sistem yang tangguh / canggih namun pada mudahnya sistem tersebut digunakan.

Hasil software LOGO tersebut mempengaruhi pemikiran Alan Kay yang mempunyai visi bahwa komputasi di masa depan adalah penggunaan mesin berukuran kecil yang tangguh (*powerful*) yang dirancang untuk user tunggal, yang disebut *personal computers*.

Bersama dengan sekelompok peneliti dari Xerox Palo Alto Research Center (PARC), Kay memadukan lingkungan pemrograman visual yang sederhana namun tangguh, Smalltalk dengan perangkat lunak komputasi personal (*personal computing*), yang disebutnya sebagai Dynabook.

5) Sistem Window dan interface WIMP (Windows, Icons, Menus and Pointers)

Manusia mampu berpikir mengenai lebih dari satu hal pada satu waktu. Dan dalam mengerjakan tugasnya, manusia sering menginterupsi pekerjaannya dan mengerjakan pekerjaan lain yang berkaitan. Jika personal computer dibuat dengan mengharuskan usernya mengerjakan pekerjaan dalam urutan yang tidak bisa dialihkan dari awal hingga selesai maka hal tersebut tidak pola kerja manusia yang telah disebutkan sebelumnya. Maka agar komputer dapat menjadi rekan kerja yang efektif, harus dibuat fleksibel untuk berganti topik seperti halnya manusia.

Karena user terlibat dalam berbagai tugas dalam satu waktu tertentu, menjadi sulit bagi komputer untuk menjaga status pekerjaan (*threads*) yang overlapping.

Perlu dipisahkan berdasarkan konteks masing-masing *threads* dan dialognya sehingga user dapat membedakannya.

Salah satu mekanisme presentasi untuk membagi dialog adalah dengan memisahkan secara fisik presentasi threads logik percakapan user-komputer yang berbeda pada layar yang disebut sebagai *window*. Dan kini pada sistem window ini, semakin banyak digunakan WIMP (Window, Icon, Menu, Pointer) interface.

6) Metapora (Metaphor)

Metapora telah cukup sukses digunakan untuk mengajarkan konsep baru dengan terminologi yang telah dipahami sebelumnya. Dan mekanisme pengajaran ini digunakan untuk memperkenalkan peralatan komputer yang relatif memiliki tehnik interaksi yang berbeda dengan peralatan yang telah ada.

Contoh dalam domain personal komputing adalah spreadsheeet yang merupakan metapora dari model akuntansi dan keuangan, kemudian ada keyboard yang merupakan metapora dari mesin ketik manual.

Namun tidak selalu semua pekerjaan yang dilakukan dengan komputer dapat diasosiasikan dengan keadaan dunia nyata

7) Manipulasi Langsung (Direct Manipulation)

Pada awal tahun 1980-an, dengan harga hardware grafik yang memiliki kemampuan dan kualitas yang tinggi menurun, para perancang mulai menyadari bahwa aplikasinya akan meningkat popularitasnya seiring dengan bertambahnya fungsi visualisasi. Pada interaksi *command-line* standar, satu-satunya cara untuk mendapatkan hasil interaksi sebelumnya adalah dengan bertanya menggunakan perintah (*command*) dan harus tahu bagaimana memberikan perintah tersebut.

Contoh lain dari direct manipulation adalah konsep WYSIWYG (*what you see is what you get*). Apa yang user lihat pada layar display pada saat menggunakan word processing misalnya, adalah bukan dokumen sebenarnya yang nantinya dihasilkan pada tahap akhir. Namun merupakan representasi atau rendering dari bagaimana rupa dokumen final nantinya.

8) Bahasa vs. Aksi (Language versus Action)

Gambaran bentuk komunikasi dari *direct manipulation* adalah interface menggantikan sistem yang berada didalamnya sehingga user tidak perlu memahami artinya pada level yang lebih rendah yaitu level sistem.

Bentuk lain adalah interface sebagai mediator antara user dan sistem. User memberikan instruksi kepada interface (berupa bahasa user)dan menjadi tanggung jawab interface untuk menjamin terlaksananya instruksi tersebut. Komunikasi seperti ini menggunakan mekanisme *indirect language*.

9) Computer-Supported Cooperative Work (CSCW)

Perkembangan komputasi lain pada tahun 1960-an adalah jaringan komputer yang memungkinkan komunikasi antara beberapa mesin (*personal computer*) yang terpisah dalam satu kesatuan grup.

Dengan adanya jaringan komputer ini, komputer personal tetap mampu bekerja secara individu dan dapat berhubungan dengan komputer lain di lingkungan kerjanya bahkan dengan seluruh dunia.

Keadaan ini memunculkan perlunya kolaborasi antar individu melalui komputer yang dikenal sebagai *Computer-Supported Cooperative Work* (CSCW).

Prinsip Yang Mendukung Pendayagunaan

Pada bagian ini dibahas prinsip umum yang dapat diaplikasikan pada rancangan sistem interaktif untuk meningkatkan daya gunanya. Prinsip ini terdiri dari tiga kategori utama, yaitu :

- ***Learnability*** :

  Learnability menyangkut fitur sistem interaktif memungkinkan user baru memahami bagaimana menggunakannya pada saat awal dan mempertahankan kinerja pada level yang maksimal.

- ***Flexibility*** :

  Flexibility berkaitan dengan banyaknya cara yang dapat ditempuh oleh end-user untuk bertukar informasi atau berkomuniaksi dengan sistem.

- ***Robustness*:**

User menggunakan komputer untuk mencapai sekumpulan tujuan yang terkait dengan pekerjaannya atau area tugas tertentu. Fitur robustness dari sebuah interaksi meliputi hal-hal yang mendukung keberhasilan pencapaian dan penilaian pencapaian tujuan tersebut atau tingkat dukungan yang diberi-kan agar user dapat menentukan keberhasil-annya atau tujuan (goal) yang diinginkan.

PROSES DESAIN

Tujuan perancangan adalah memberikan tehnik yang dapat dihandalkan untuk perancangan secara berulang dari sistem interaktif yang sukses dan berdaya guna. Di ilmu komputer, terdapat sebuah sub disiplin besar yang membahas isu manajemen dan tehnik dari pengembangan software yang dikenal sebagai rekayasa perangkat lunak (*software engineering*).

Salah satu hal dasar dalam rekayasa perangkat lunak adalah daur hidup perangkat lunak (*software life cycle*) yang mendeskripsikan aktifitas *IMK*

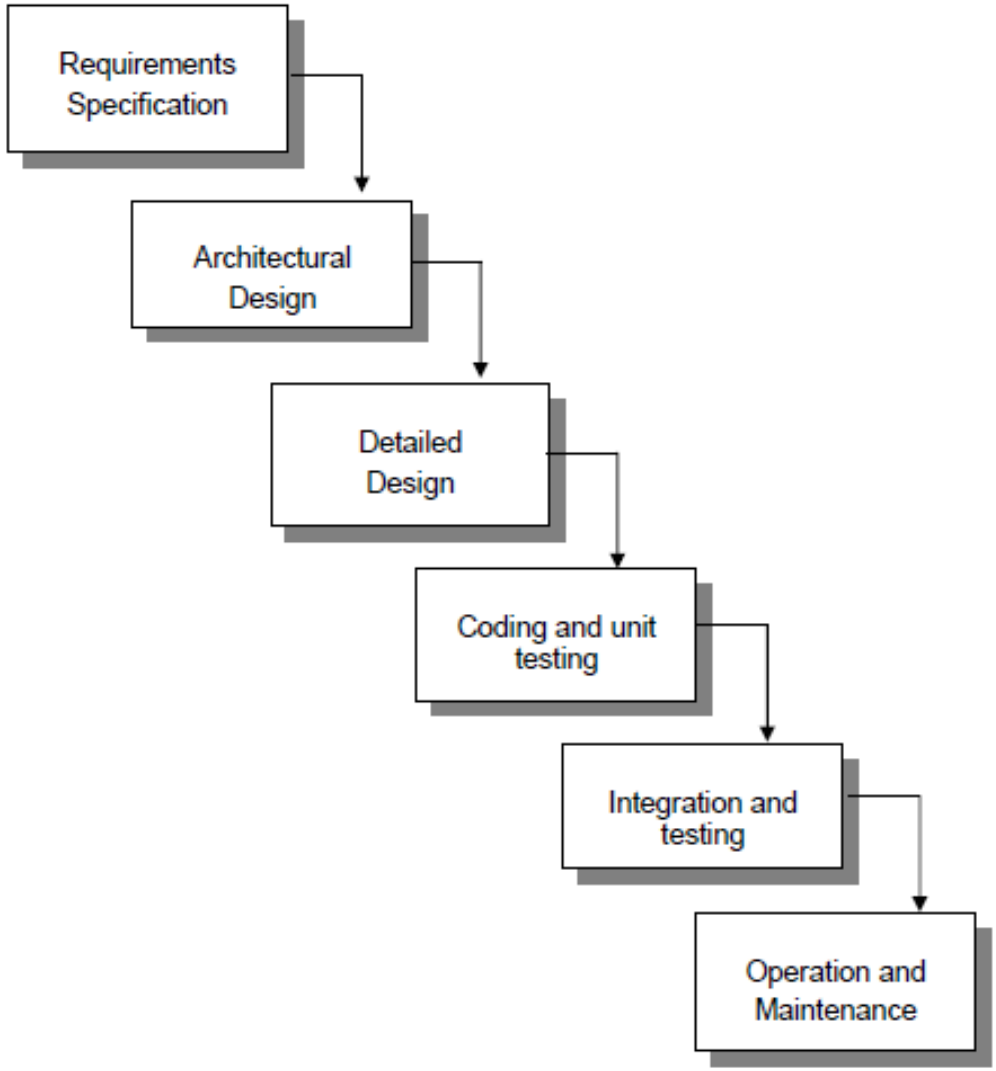
Sofware Life Cycle

Software life cycle adalah sebuah usaha untuk mengidentifikasi aktifitas yang terjasi selama pengembangan sebuah perangkat lunak. Aktifitas ini kemudian diurutkan sesuai dengan waktu pelaksanaannya pada proyek pengembangan manapun dan diaplikasikan tehnik yang tepat pada setiap aktifitasnya.

Aktifitas Pada Life Cycle

Aktifitas life cycle direpresentasikan dalam grafik pada gambar 3.1 berikut ini.

Bagan ini dikenal sebagai model waterfall karena mengikuti bentuk air terjun dengan satu aktifitas menuju ke aktifitas berikutnya.



Gambar 3.1 Aktifitas Pada Siklus Hidup Software Model Waterfall

Requirement Specification

Pada tahap requirement specification, desainer dan customer mencoba menangkap deskripsi seperti apa nantinya sistem yang sebenarnya akan dibangun. Aktifitas ini melibatkan pencarian informasi dari customer mengenai lingkungan kerja tempat sistem ini nantinya akan diimplementasikan.

Architectural Design

Aktifitas ini memfokuskan pada bagaimana sistem menyediakan layanan seperti yang diharapkan. Aktifitas pertama adalah *high-level decomposition* yang membagi sistem menjadi komponen-komponen sesuai dengan fungsinya. Pembagian ini dapat didasarkan pada pembagian yang sudah ada di sistem yang lama atau membuat dari baru.

Detailed Design

Architectural design menghasilkan dekomposisi deskripsi sistem yang memungkinkan pengembangan komponen secara terpisah untuk kemudian diintegrasikan kembali nantinya. Agar dapat diimplementasikan dengan bahasa pemrograman, desainer harus melengkapi deskripsi tersebut dengan deskripsi yang lebih detail. Oleh karena itu, tahap *detailed design* adalah perbaikan dari deskripsi komponen yang dihasilkan oleh *architectural design*.

Coding and Unit Testing

Hasil dari *detailed design* harus dalam bentuk yang dapat diimplementasikan ke executable programming language. Setelah coding, setiap komponen diuji untuk memverifikasi apakah berjalan dengan benar sesuai dengan kriteria yang yang telah ditetapkan pada tahap-tahap awal.

Integration and Testing

Setelah komponen-komponen diimplementasikan dan diuji secara individual, maka komponen tersebut harus diintegrasikan seperti yang dideskripsikan pada architectural design. Pengujian lebih lanjut dilakukan untuk memastikan perilaku yang benar dan tidak ada konflik penggunaan sumber daya bersama.

Maintenance

Setelah produk di-release, semua pekerjaan yang dilakukan terhadap sistem dianggap sebagai pemeliharaan (*maintenanc*e) sampai produk memerlukan desain ulang menjadi versi baru atau produk tidak terpakai lagi. Maintenance melibatkan koreksi terhadap kesalahan / errror yang ditemui pada sistem setelah di-release dan dilakukan perbaikan terhadap sistem.

Validasi dan Verifikasi

Selama life cycle, rancangan harus dicek untuk memastikan produk memenuhi kebutuhan customer (*high-level requirement*), lengkap, dan konsisten. Proses pengecekan ini disebut sebagai validasi dan verifikasi.

Aturan Perancangan

Salah satu masalah pada proses perancangan berpusat pada user adalah bagaimana membuat desainer memiliki kemampuan untuk menentukan konsekuensi terhadap usability dari keputusan perancangan yang mereka ambil. Maka dibutuhkan aturan perancangan (*design rules*) yang dapat diikuti untuk meningkatkan usability dari produk software yang dibangun.

Rekayasa Daya Guna (*Usability Engineering*)

Pendekatan lain pada perancangan yang berpusat pada user adalah penetapan tujuan rekayasa daya guna (*usability engineering*) pada proses perancangan. Proses rekayasa melibatkan interpretasi terhadap arti secara bersama, tujuan yang disetujui bersama, dan pemahaman mengenai bagaimana mengukur pencapaian kepuasan. Penekanan *usability engineering* adalah mengetahui dengan pasti kriteria apa yang akan digunakan untuk menilai kegunaan produk. Pengujian *usability* suatu produk didasarkan pada pengukuran pengalaman user dengan produk tersebut.

Desain Iteratif dan Prototyping

Seperti yang telah dikemukakan di depan bahwa spesifikasi kebutuhan sistem interaksi tidak dapat dilengkapi di awal life cycle. Satu-satunya cara untuk memastikan tercakupnya fitur-fitur yang potensial adalah dengan membangunnya kemudian dites pada user. Kesalahan desain yang ditemukan pada saat testing kemudian dikoreksi. Inilah inti dari desain iteratif.

Desain (*Design Rationale*)

Dalam merancang sistem komputer manapun, diambil keputusan-keputusan yang terkait dengan perancangan untuk mengakomodasi kebutuhan user ke dalam sistem. Kadangkala sulit untuk mengungkapkan kembali alasan atau rasionalitas yang melandasi keputusan-keputusan tersebut.

Rasionalitas desain (*design rationale*) adalah informasi yang menjelaskan alasan mengapa suatu keputusan dalam suatu tahap perancangan / desain sistem komputer dibuat atau diambil, termasuk deskripsi struktural atau arsitektural dan deskripsi fungsi atau perilakunya.

Beberapa keuntungan rasionalitas desain :

- Dalam bentuk yang eksplisit, rasionalitas desain menyediakan mekanisme komunikasi di antara anggota tim desain sehingga pada tahapan desain dan atau pemeliharaan (*maintenance*).
- Akumulasi pengetahuan dalam bentuk rasionalitas desain untuk suatu set produk dapat digunakan kembali untuk mentransfer hal yang berhasil dalam suatu situasi ke situasi lainnya yang mirip.

- Usaha yang diperlukan untuk menghasilkan sebuah rasionalitas desain memaksa desainer untuk bersikap hati-hati dalam mengambil suatu keputusan desain.

Rasionalitas Desain Beorientasi Proses (*Process-oriented Design Rationale*)

Sebagian besar rasionalitas desain mengadaptasi bentuk ***issue-based information system*** (IBIS) yang representasikan dialog perencanaan dan desain dan dikembangakan pada tahun 1970-an oleh Rittel. IBIS dibuat dalam bentuk hirarki, ***issue*** sebagai akar dan merepresentasikan masalah utama atau pertanyaan yang dituju oleh *argument*. Berbagai ***position*** dihubungkan secara langsung dengan *issue* sebagai solusi potensial. Masing-masing position ditunjang oleh ***argument***. Hirarki ini dapat berkembang ke level berikutnya dengan berkembanganya issue menjadi beberapa ***sub-issue***.

Analisis Ruang Desain (*Design Space Analysis*)

MacLean dan rekan-rekannya mengembangkan pendekatan rasionalitas desain yang lebih detail dengan menekankan pada struktur post hoc dari ruang alternatif desain yang muncul pada proyek perancangan. Pendekatan ini disusun dalam bentuk *Question*, *Option*, dan *Criteria* (QOC) dan disebut sebagai *design space analysis*.

Rasionalitas Desain Psikologis (*Psychological Design Rationale*)

Kategori rasionalitas desain yang terakhir adalah *psychological design rationale* yang mencantumkan secara eksplisit aspek psikologis dari usability sistem interaktif untuk membuat produk yang sesuai dengan tugas yang dilakukan user.

Psychological design rationale bertujuan untuk menunjukkan konsekuensi dari desain terhadap tugas yang dilakukan user.

Tahap awal dari *psychological design rationale* adalah mengidentifikasi tugas yang akan dilayani oleh sistem dan mengkarakteristikan tugas tersebut dalam pertanyaan user dalam rangka mengerjakan tugas tersebut.

Dengan membuat *psychological design rationale* diharapkan desainer akan semakin memperhatikan sifat tugas yang dilakukan user dan memanfaatkan konsekuensi suatu desain untuk memperbaiki rancangan berikutnya.

TEKNIK EVALUASI	→ Evaluasi pengujian dari sistem interaktif secara fungsional → Pengambilan evaluasi → Beberapa pendekatan desain evaluasi → Beberapa pendekatan untuk mengimplementasikan evaluasi.