PERKULIAHAN KE 3

Tujuan Instruksional Khusus (TIK)
Mahasiswa mampu :
- Menjelaskan prinsip dan paradigma interaksi
- Mengenali siklus pengembangan software
- Menjelaskan aturan desain, rekayasa, dan rasionalitas desain
- Menjelaskan prototipe desain

Pokok Bahasan :

- Pendahuluan
- Paradigma Interaksi
- Prinsip2 Pendukung Interaksi
- Siklus Perkembangan Software
- Aturan Desain
- Rekayasa Kegunaan (*Usability Engineering*)
- Desain Iteraktif dan Prototipe
- Desain Rasionalitas

Deskripsi Singkat : bahasan ini tentang paradigma yang memaparkan model-model interaksi yang sudah ada dan dianggap berhasil serta prinsip interaksi yang sebaiknya terdapat dalam suatu desain sistem interaksi. Dibahas pula proses perancangan sebuah sistem interaksi dan aspek-aspeknya seperti siklus pengembangan, aturan desain, rekayasa kegunaan, prototipe dan rasionalitas desain.

Bahan Bacaan : Dix, Alan et.al, ***HUMAN-COMPUTER INTERACTION***, Prentice Hall, Europe, 1993, hal 9-114

Johnson, P., ***HUMAN-COMPUTER INTERACTION : Psychology, Task Analysis and Software Engineering***, McGraw-Hill, England UK, 1992

Sutcliffe, A. G., ***HUMAN-COMPUTER INTERFACE DESIGN***, 2ND Edition, MacMillan, London, 1995

PARADIGMA, PRINSIP INTERAKSI, DAN PROSES DESAIN

Tinjauan

- Tujuan dari perancangan sistem interaksi adalah daya guna (*usability*) yang maksimum.
- Beberapa contoh strategi pembangunan sistem interaksi yang efektif memberikan paradigma bagi perancangan sistem interaksi yang berdaya guna. Evolusi paradigma ini juga memberikan perspektif yang baik dalam sejarah komputasi.
- Prinsip yang lebih abstrak menawarkan pemahaman mengenai daya guna dalam lingkup yang umum.
- Rekayasa perangkat lunak (*software engineering*) menyediakan alat pemahaman tentang struktur proses desain, dan proses tersebut dapt dinilai keefektifannya dalam perancangan sistem interaksi.
- Aturan desain dalam bentuk standar dan *guideline* memberikan arah bagi perancangan dalam cakupan yang umum dan kongkret, untuk meningkatkan sifat interaktif dari sistem.
- Rekayasa daya guna (*usability engineering*) mendorong penggunaan kriteria penilaian keberhasilan sebuah produk dari sisi keberdayagunaan secara eksplisit.
- Desain iteratif berusaha memadukan umpan balik yang penting dari user pada tahap awal perancangan untuk mengambil keputusan krusial yang mempengaruhi daya guna.
- Perancangan melibatkan pengambilan keputusan diantara sekian banyak alternatif. Rasionalitas desain memberikan media eksplisit untuk merekam keputusan serta konteks pengambilan keputusan perancangan.

PARADIGMA DAN PRINSIP INTERAKSI

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 (*usable*). 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 (*usability*) 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.

Kemajuan yang sangat pesat dari teknologi komputer telah meningkatkan kemampuan mesin komputer dan memperbesar bandwith komunikasi dengan manusia. Namun kemajuan teknologi tersebut tidak cukup untuk meningkatkan dayagunanya. Kuncinya terletak pada kreatifitas pemanfaatan teknologi komputer pada aplikasi tertentu untuk meningkatkan kemampuan manusia.

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.

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.

Pada konsep *time-sharing*, komputer diperuntukan bagi individual user dan peningkatan keluaran (*throughput*) sistem menjadikan user lebih reaktif dan kolaboratif. Dapat dikatakan bahwa *time-sharing* memungkinkan interaksi interaktif antara manusia dengan komputer.

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

sama. Sehingga dengan adanya "*Sketchpad* " ini interaksi antara manusia dengan komputer menjadi lebih baik dengan informasi yang dihasilkan oleh komputer menjadi lebih mudah dipahami oleh manusia / user.

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. Douglas Engelbart, sesorang lulusan UCLA Berkeley, berpendapat bahwa dengan meningkatkan kemampuan manusia, berarti bertambah pula kapabilitas manusia untuk memecahkan masalah yang kompleks. 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.

Komputer Pribadi (Personal Computing)

Programming toolkit telah menjadi alat bagi mereka yang memiliki kemampuan komputasi atau para pemrogram untuk meningkatkan produktivitasnya. Namun Engelbart mempunyai visi bahwa komputer tidak hanya diperuntukkan bagi mereka yang mengerti komputer (computer literate) 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.

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.

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.

The Xerox Alto and Star adalah workstation pertama yang menggunakan metaphor dari office desktop. Sebagian besar tugas manajemen terkait dengan manipulasi file. Dengan mengaitkan tugas-tugas manipulasi file tersebut dengan lingkungan kerja di kantor membuat pekerjaan dengan komputer tersebut menjadi mudah. Contoh lain 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. Dan hal ini dapat menjadi masalah. Namun terlepas dari hal tersebut, kini sukses secara komersial telah diraih dari penggunaan metaphor ini, seperti yang kita lihat pada windows, menu, button, icon, dan pallete.

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.

Dengan adanya umpan balik (*feedback*) atau respon cepat secara visual dan audio pada layar dengan resolusi tinggi dan sistem suara berkualitas akan memudahkan pemberian informasi mengenai setiap aksi user yang dieksekusi. Dan tehnik ini dikenal sebagai *direct manipulation* (manipulasi langsung). Sukses komersial pertama yang mendemonstrasikan *direct manipulation* ini adalah personal computer macintosh dari Apple Computer Inc. Manipulasi langsung ini memungkinkan user untuk mengubah keadaan internal sistem dengan cepat.

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. Implikasi dari WYSIWYG ini adalah perbedaan antara representasi dan hasil akhir adalah minimal, dan user dapat dengan mudah memvisualisasikan hasil akhir dari representasi yang diberikan komputer.

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 dan menjadi tanggung jawab interface untuk menjamin terlaksananya instruksi tersebut. Komunikasi seperti ini menggunakan mekanisme *indirect language*.

Terdapat dua interpretasi dari bentuk komunikasi ini, **pertama,** user diharuskan mengerti keadaan fungsi sistem dan interface sebagai mediator tidak perlu terlalu banyak melakukan penerjemahan, dalam hal ini berarti kembali lagi seperti keadaan sebelum adanya *direct manipulation*; **kedua,** user tidak perlu memahami keadaan fungsi sistem dan interface menjalankan peran yang aktif untuk menerjemahkan operasi yang diinginkan oleh user menjadi operasi sistem. Contoh model yang kedua

adalah pada sistem pencarian informasi (*information retrieval system*). Kita tidak perlu tahu bagaimana informasi diorganisasikan, pencarian dilakukan dengan pertanyaan yang ada pada konsep user.

Paradigma bahasa (*language*) memiliki kelebihan dan kekurangan dibandingkan dengan paradigme aksi (*action*). Pada paradigma aksi, lebih mudah melakukan tugas yang sederhana tanpa adanya resiko melakukan kesalahan, sebagai contoh dengan mengenali dan menunjuk obyek secara langsung mengurangi kesulitan identifikasi dan kemungkinan misidentifikasi. Namun pada pekerjaan yang kompleks, paradigma aksi lebih rumit untuk melakukannya karena membutuhkan pengulangan prosedur yang sama dengan hanya sedikit modifikasi. Pada paradigma bahasa, dimungkinkan untuk mendeskripsikan prosedur generik misalnya mekanisme looping, sekali saja dan dapat dijalankan tanpa intervensi lebih jauh dari user.

Hypertext

Pada tahun 1945, Vannevar Bush mengkoordinasi 6000 ilmuwan merasa kesulitan besar dari penelitian yang dilakukan saat itu adalah sulit untuk mendapatkan literatur yang terus bertambah. Saat itu, paper diorganisasikan dalam bentuk linear, dan kadangkala ada bagian dari paper tersebut yang menyebabkab pembaca perlu menggali lebih dalam lagi. Penyimpanan informasi dalam format linear ini tidak banyak mendukung pengaksesan informasi secara random dan browsing asosiatif.

Kemudian dia membangun sebuah inovasi dalam penyimpanan informasi dan mekanisme pengambilannya yang disebut sebagai *memex* yang bertujuan untuk meningkatkan kemampuan menyimpan dan mengambil informasi dengan link asosiasi random. *Memex* ini intinya adalah sebuah desk yang mampu memproduksi dan menyimpan salinan fotografik dari dokumen informasi dalam jumlah besar dan dapat menyimpan trak dari link dari dokumen yang berbeda.

Hingga akhrnya, pada pertengahan tahun 1960-an, Ted Nelson memberikan istilah *Hypertext* bagi metode penyimpanan informasi dalam format non-linear yang memungkinkan akses atau browsing secara non-linear atau random ini.

Multi-Modality

Sistem interaktif multi-modality adalah sistem yang tergantung pada penggunaan beberapa (*multiple*) saluran (*channel*) komunikasi pada manusia. Dengan definisi ini, semua sistem interaktif dapat dianggap sebagai sistem muti-modality, karena manusia selalu menggunakan saluran / indera visual dan haptic pada saat

memanipulasi komputer. Bahkan kita sering menggunakan saluran audio untuk mendengarkan apakah komputer benar beroperasi dengan semestinya.

Sistem multi-modality modern sangat besar melibatkan penggunaan banyak (*multiple*) saluran komunikasi secara simultan baik untuk input maupun output. Normalnya, manusia memproses informasi menggunakan beberapa saluran komunikasi secara simultan. Para perancang sistem ini mencoba meniru fleksibilitas observasi dan artikulasi yang dimiliki oleh manusia dengan meningkatkan kemampuan ekspresi input dan output pada sistem interaktif.

Multi-modal, multi-media, dan virtual reality adalah contoh dari penelitian dalam bidang sistem interaktif yang dikategorikan dalam bidang sistem multi sensor (*multi-sensory system*).

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).

Perbedaan utama antara sistem CSCW dengan sistem interaksi individual adalah tidak dapat diabaikannya aspek sosial kelompok dari user yang tergabung. Sistem CSCW dibangun untuk memungkinkan interaksi antara user melalui komputer sehingga kebutuhan sekian banyak user tersebut harus terpenuhi dalam satu produk. Salah satu contoh sistem CSCW ini adalah *electronic mail* (*email*). Email merupakan sistem CSCW yang bersifat asynchronous yang tidak mengharuskan user bekerja pada waktu yang bersamaan. Penerima mail tidak harus membuka suratnya pada waktu yang sama dengan terkirimnya surat. Sebaliknya sistem CSCW synchronous membutuhkan partisipasi simultan dari para usernya. Materi CSCW ini akan dibahas pada bab tersendiri.

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*** : kemudahan yang memungkin-kan user baru berinteraksi secara efektif dan dapat mencapai performance yang maksimal
- ***Flexibility*** : menyediakan banyak cara bagi user dan sistem untuk bertukar informasi
- ***Robustness***: tingkat dukungan yang diberi-kan agar user dapat menentukan keberhasil-annya atau tujuan (goal) yang diinginkan.

Learnability

Learnability menyangkut fitur sistem interaktif memungkinkan user baru memahami bagaimana menggunakannya pada saat awal dan mempertahankan kinerja pada level yang maksimal. Berikut ini adalah prinsip-prinsip yang mendukung learnability.

Tabel 3.1 Prinsip yang Mempengaruhi Kemampuan Belajar (Learnability)

Prinsip	Definisi	Prinsip yang Terkait
Predictability	Mendukung user untuk menentukan efek dari aksi selanjutnya / 'future action' berdasarkan catatan / sejarah interaksi sebelumnya	Operation visibility
Synthesizability	Mendukung user untuk memperkirakan efek dari operasi sebelumnya pada keadaan saat ini	Immediate/ Eventual Honesty
Familiarity	Pengetahuan dan pengalaman user dalam domain berbasis komputer atau dunia nyata lainnya dapat diterapkan ketika berinteraksi dengan sistem yang baru	Guessability Affordance

Generalizability	Mendukung user untuk menambah pengetahuan dari interaksi spesifik di dalam dan di luar aplikasi aplikasi ke situasi lainnya yang mirip	
Consistency	Kemiripan dalam perilaku input / output yang muncul dari situasi atau tugas obyektif yang sama	

Flexibility

Flexibility berkaitan dengan banyaknya cara yang dapat ditempuh oleh end-user untuk bertukar informasi atau berkomuniaksi dengan sistem. Terdapat beberapa aspek yang berkontribusi pada sifat fleksibilitas interaksi seperti yang diganbarkan pada tabel berikut ini .

Tabel 3.2 Prinsip yang Mempengaruhi Fleksibilitas

Prinsip	Definisi	Prinsip yang Terkait
Dialogue Initiative	Memungkinkan user terbebas dari kendala-kendala buatan (*artificial*) pada dialog input yang dipaksakan oleh sistem	System / User preemtiveness
Multi-Treading	Kemampuan system untuk mendukung interaksi user yang berhubungan dengan lebih dari satu task pada suatu saat / waktu	Concurrent vs. interleaving, multi-modality
Task Migratability	Kemampuan untuk melewatkan / memberikan kontrol dari eksekusi task yang diberikan sehingga menjadi task internal user atau sistem atau berbagi antara keduanya	
Substitutivity	Memungkinkan nilai-nilai (*values*) ekuivalen antara input dan output yang masing-masing secara bebas dapat disubstitusi	Representasi perkalian, kesamaan kesempatan (opportunity)
Customizability	Kemampuan user interface untuk dimodifikasi oleh user atau system	Adaptivity, Adaptability

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, seperti yang tercantum pada tabel di bawah ini.

Tabel 3.3 Prinsip yang Mempengaruhi Robustness

Prinsip	Definisi	Prinsip yang Terkait
Observability	Kemampuan user untuk mengevaluasi keadaan internal system dari representasi yang dapat dimengerti / dirasakan	Browsability, static / dynamic defaults, reachability, persistence, operation visibility
Recoverability	Kemampuan user untuk melakukan koreksi bila sebuah error (kesalahan) telah dikenali	Reachability, forward / backward recovery commensurate effort
Responsiveness	Bagaimana user mengetahui / menyadari laju komunikasi dengan sistem	Stability
Task Conformance	Tingkatan dimana sistem pelayanan mendukung semua tasks yang user ingin lakukan dan dengan cara yang user ketahui	Task completeness, task adequacy

PROSES DESAIN

Pendahuluan

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

yang terjadi mulai dari pembentukan konsep awal hingga tahap penggantian sistem dan implementasi.

Isu interaksi manusia dan komputer yang menyangkut daya guna (*usability*) sistem interaktif relevan dengan seluruh aktifitas pada software life cycle. Sehingga software engineering untuk sistem interaktif bukan semata-mata menambahkan sebuah tahapan pada software life cycle namum lebih pada melibatkan tehnik yang berada sepanjang software life cycle.

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.

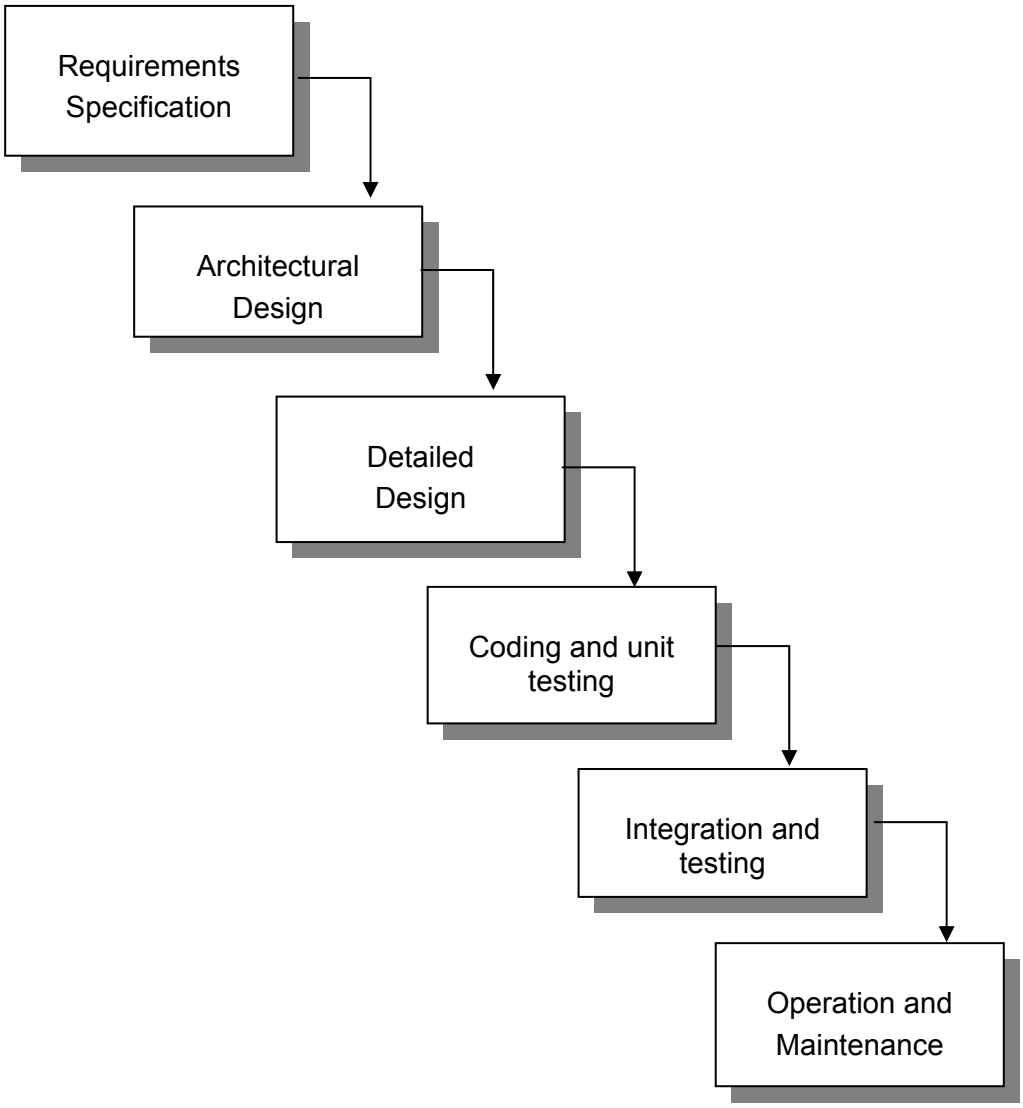
Pada pengembangan produk perangkat lunak, kita memperhatikan dua buah pihak, yaitu pelanggan (*customer*) yang akan menggunakan produk dan desainer yang menghasilkan produk. Umumnya pelanggan dan desainer adalah sekelompok orang, dan pada beberapa hal customer dapat menjadi desainer sekaligus. Kadang penting untuk membedakan customer yang memberikan kerja atau menjadi klien bagi desainer dengan customer yang merupakan user yang benar-benar akan menjalankan sistem. Kedua peran tersebut dapat dipegang oleh orang atau kelompok orang yang berbeda.

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.

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.



Gambar 3.1 Aktifitas Pada Siklus Hidup Software Model Waterfall

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. *Architectural design* tidak hanya meliputi pembagian fungsi sistem yang nantinya akan menyediakan layanan, namun juga mendeskripsikan keterhubungan dan pemakaian bersama sumber daya antara komponen tersebut.

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*. Perilaku yang ditunjukkan oleh deskripsi pada level di atasnya, harus terdapat pula di deskripsi detailnya.

Coding and Unit Testing

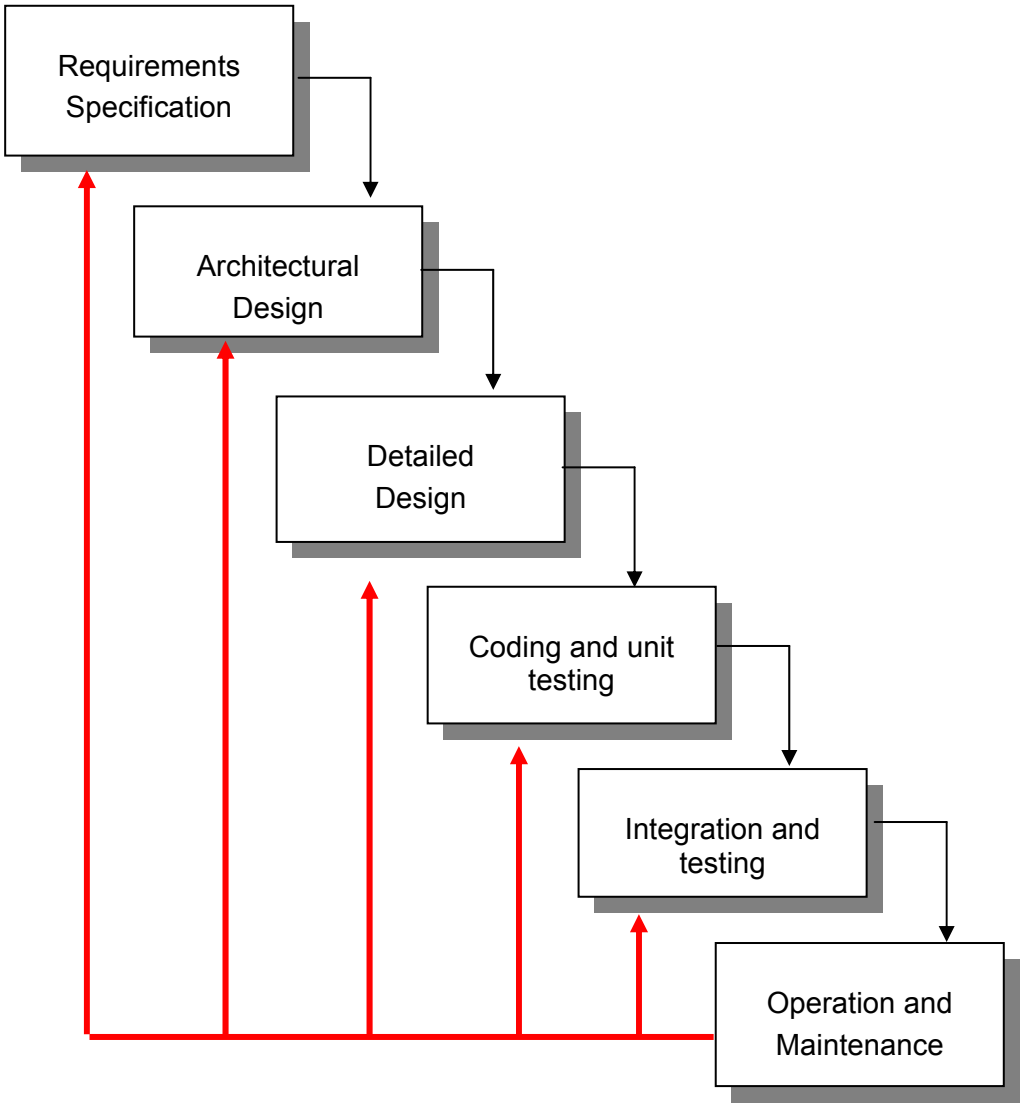
Hasil dari *detailed design* harus dalam bentuk yang dapat diimplementasikan ke executable programming language. Setelah coding, setiap komponen diuji untuk memverifikasi apakah berjalan dengan benar sesuai dengan kriteria yang yang telah ditetapkan pada tahap-tahap awal.

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. Pada tahap ini juga dimungkinkan untuk melakukan tes (*acceptance test*) dengan customer untuk memastikan sistem yang dibuat memenuhi kebutuhan mereka. Setelah *acceptance test* maka produk dapat di-*release* kepada customer.

Maintenance

Setelah produk di-release, semua pekerjaan yang dilakukan terhadap sistem dianggap sebagai pemeliharaan (*maintenance*) 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. Sehingga tahap maintenance memberikan feedback pada semua aktifitas lain pada life cyle, seperti yang ditunjukkan pada gambar 3.2



Gambar 3.2 Feedback dari Maintenance ke Aktifitas Perancangan Lainnya

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. Boehm memberikan definisi yang membedakan validasi sebagai *desigining "the right thing"*, dan verifikasi sebagai *designing "the thing right"*.

Verifikasi dari suatu desain umumnya akan terjadi pada satu aktifitas life cycle atau antara dua aktifitas yang berurutan sedangkan validasi dilakukan pada berbagai aktifitas yang membutuhkan kepuasan customer. Validasi lebih bersifat subyektif dibandingkan verifikasi. Hal ini utamanya disebabkan karena adanya perbedaan antara bentuk bahasa deskripsi kebutuhan (*requirement*) dengan bahasa

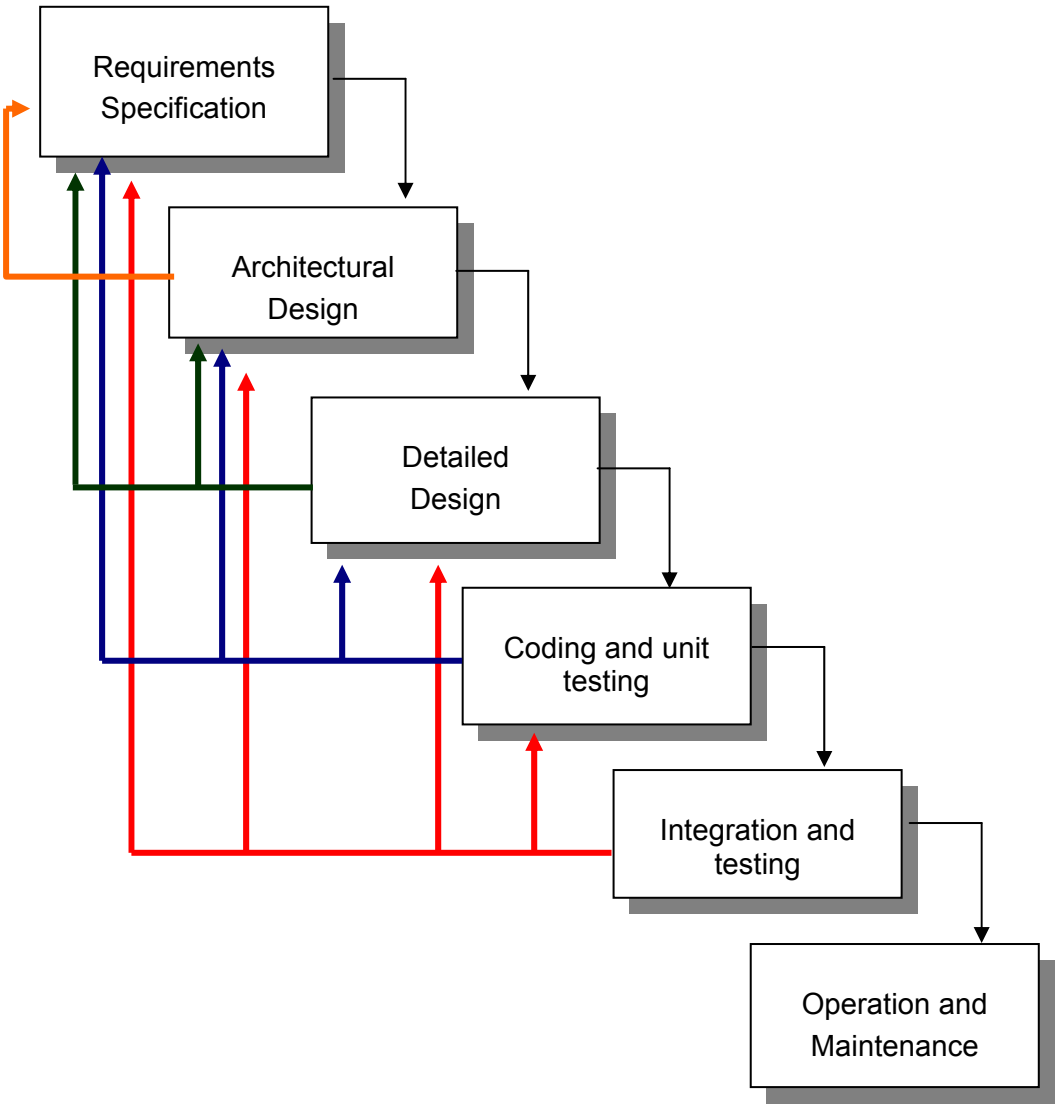
perancangan. Pada bidang IMK, validasi ini sering disebut sebagai evaluasi yang dapat dilakukan secara terpisah oleh desainer atau bekerja sama dengan user.

Sistem Interaktif dan Software Life Cycle

Software life cycle tradisional muncul pada tahun 1960-an dan 1970-an untuk menyediakan struktur bagi pengembangan sistem software besar. Saat itu mayoritas dari sistem-sistem besar tersebut merupakan aplikasi pemrosesan data bisnis. Dan sistem-sistem tersebut tidak bersifat interaktif melainkan merupakan sistem dengan pemrosesan batch. Seiring dengan perkembangan komputer personal, kini sistem cenderung semakin interaktif

Life cycle yang diterangkan sebelumnya mempresentasikan proses perancangan dalam urutan dari atas ke bawah. Dalam kenyataannya, meskipun sistem pemrosesan bacth, proses perancangannya dilakukan secara iteratif, yaitu pekerjaan yang dilakukan pada satu aktifitas mempengaruhi aktifitas sebelum dan setelahnya pada siklus pengembangan, seperti yang tergambar pada gambar 3.3.

Software life cycle tradisional cocok dengan pendekatan prinsipal terhadap proses perancangannya, yaitu jika kita mengetahui dari awal apa yang akan kita bangun maka kita dapat menjalani perancangan dengan struktur yang terurut untuk mencapai tujuan yang ditetapkan. Namun dalam prakteknya tidak semua kebutuhan user dapat kita daftar pada tahap awal mulainya pembangunan sistem. Oleh karena itu seperti pada gambar 3.3, penemuan fakta pada suatu tahap dapat membawa tahap tersebut beriterasi ke tahap sebelumnya. Terlebih lagi pada sistem interaktif, tidak semua kebutuhan / persyaratan sistem (*system requirement*) dapat diperoleh pada tahap awal. Sehingga sistem harus dibangun dengan berinteraksi dengan user, diobservasi dan dievaluasi untuk meningkatkan daya guna (*usability*) sistem tersebut. Hasil evaluasi ini dapat menjadi masukan untuk proses iterasi perancangan ke tahap sebelumnya.



Gambar 3.3 Representasi Iterasi Pada Software Life Cycle Model Waterfall

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. Kita dapat mengklasifikasikan aturan tersebut berdasarkan dua dimensi yaitu berdasarkan autoritas (*authority*) dan generalitasnya (*generality*). Berdasarkan autoritas mengindikasikan apakah aturan tersebut harus diikuti atau disarankan dalam suatu proses perancangan. Berdasarkan generalitas menunjukkan apakah aturan tersebut dapat diaplikasikan pada semua situasi

perancangan atau hanya terbatas pada situasi perancangan tertentu. Terdapat dua jenis aturan yang terkait dengan keterangan diatas, yaitu *standard* dan *guideline.* Secara umum, **standard** memiliki autoritas yang tinggi namun terbatas pada pengimplementasiannya. Sedangkan **guideline** cenderung memiliki autoritas yang rendah namun lebih banyak (umum) pengimplementasiannya.

Aturan desain sistem interaktif dapat didukung oleh disiplin ilmu psikologi, kognitif, ergonomi, sosiologi, ekonomi maupun teori komputasi.

Standar

Standar bagi sistem interaktif umumnya diatur oelh badan nasional atau internasional untuk menjamin penerimaan aturan tersebut oleh sekelompok besar komunitas. Standar sistem interaktif dapat dibuat untuk bidang hardware maupun software. Ada dua karakteristik yang membedakan standar untuk hardware dengan standar software, yaitu :

1. **Teori yang mendasari**, standar hardware didasarkan pada pemahaman terhadap psikologi, ergonomi, dan hasilnya relatif bersifat tetap, sudah diketahui, dan mudah beradaptasi dengan desain hardware yang ada. Sedangkan standar software didasarkan pada pemahaman terhadap psikologi atau ilmu kognitif, dan bentuknya kurang formal, masih berkembang, dan tidak mudah diinterpretasikan pada bahasa perancangan.
2. **Perubahan**, hardware lebih sulit dan mahal untuk berubah dibandingkan software yang fleksibel.

Guideline

Ketidaklengkapan teori yang mendasari perancangan mengakibatkan sulitnya menetapkan standar yang spesifik dan autoritatif. Sebagai akibatnya, mayoritas aturan perancangan bagi sistem interaktif bersifat pemberian saran (*suggestive*) dan lebih umum. Fokus kita dalam memeriksa guideline adalah dengan menentukan kemampuan diaplikasikannya guideline tersebut dalam berbagai tahap perancangan. Guideline yang bersifat lebih abstrak, akan semakin mendekati dengan prinsip yang dibahas pada bab sebelumnya, dan akan cocok pada tahap *requirement spesification*. Guideline yang bersifat lebih spesifik, lebih sesuai untuk *detailed design*. Guideline ini juga dapat diperluas hingga batas tertentu, dan menyediakan mekanisme untuk menerjemahkan spesifikasi detailed design menjadi implementasi aktual.

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.

Terkait dengan software life cycle, satu fitur penting *usability engineering* adalah spesifikasi daya guna (*usability specification)* yang merupakan komponen-komponen interaksi user dengan sistem yang memberikan kontribusi pada usability sebuah produk. *Usability specification* dijadikan bagian dari spesifikasi kebutuhan (*requirement specification*). Berikut ini adalah contoh *usability specification* dari perancangan panel kendali (control panel) Video Cassette Recorder (VCR).

Tabel 3.4 Contoh *usability specification* untuk fungsi *undo* pada VCR

Attribute :	Backward recoverability
Measuring Concept :	Undo an erroneous programming sequence
Measuring Method :	Number of explicit user action to undo current program
Now Level :	No current product allows such an undo
Worst Case :	As many actions as it takes to program in mistake
Planned Level :	A maximum of two explicit user action
Best Case :	One explicit cancel action

Measuring concept adalah penjabaran dari atribut yang akan diukur, pada contoh di atas ***attribute*** *backward recoverability* merupakan aksi mengembalikan keadaan semula dari urutan pemrograman yang salah (*undo an erroneous programming sequence*). ***Measuring method*** menunjukkan bagaimana atribut akan diukur. ***Now level*** mengindikasikan keadaan saat ini dari sistem yang ada di pasaran. Nilai ***worst case*** adalah nilai terendah yang dapat diterima dari hasil pengukuran. ***Planned level*** adalah target perancangan sedangkan ***best case*** adalah kondisi yang dianggap sebagai hasil terbaik yang mungkin dihasilkan dari pengukuran teknologi yang ada pada saat itu.

Tabel 3.5 berikut ini menunjukkan daftar kriteria pengukuran yang dapat digunakan sebagai *measuring method*, sedangkan tabel 3.6 menunjukkan beberapa cara menentukan *worst case* serta *best case*. Pengukuran seperti yang dilakukan oleh usability engineering ini juga dikenal sebagai *usability metric*.

Tabel 3.5 Kriteria untuk *Measuring Method Usability Engneering*

1	Time to complete a task
2	Percent of task completed
3	Percent of task completed per unit time
4	Ratio of successes to failures
5	Time spent in errors
6	Percent of number of errors
7	Percent of number of competitors better than it
8	Number of commands used
9	Frequency of help and documentation use
10	Percent of favourable/unfavourable user comments
11	Number of repetition of failed commands
12	Number of runs of successes and of failures
13	Number of times interface misleads the user
14	Number of good and bad features recalled by users
15	Number of available commands not invoked
16	Number of regressive behaviours
17	Number of users preferring your system
18	Number of times users need to work around a problem
19	Number of times the user is disrupted from a work task
20	Number of times user loses control of the system
21	Number of times user expresses frustration or satisfaction

Tabel 3.6 Beberapa Cara untuk Menentukan Level Pengukuran (*worst case* serta *best case)*

Set levels with respect to information on
1. An existing system or previous version
2. Competitive system
3. Carrying out the task without use of a computer system
4. An absolute scale
5. Your own prototype
6. User's own earlier performance
7. Each component of a system separately
8. A successive split of the difference between best and worst values observed in user test

Tabel 3.7 mendeskripsikan contoh usability metric ISO 9241 yang dikelompokkan berdasarkan kontribusinya terhadap tiga kategori usability, yaitu efektifitas (*effectiveness*), efisiensi (*efficiency*), dan kepuasa (*satisfaction*).

Tabel 3.7 Contoh Usability Metric dari ISO 9241

Usability Objective	Effectiveness Measures	Efficiency Measures	Satisfaction Measures
Suitability for the task	Percentage of goals achieved	Time to complete a task	Rating scale for satisfaction
Appropriate for trained user	Number of power features used	Relative efficiency compared with an expert user	Rating scale for satisfaction with power features
Learnability	Percentage of functions learned	Time to learn criterion	Rating scale for ease learning
Error tolerance	Percentage of errors corrected successfully	Time spent on correcting errors	Rating scale for error handling

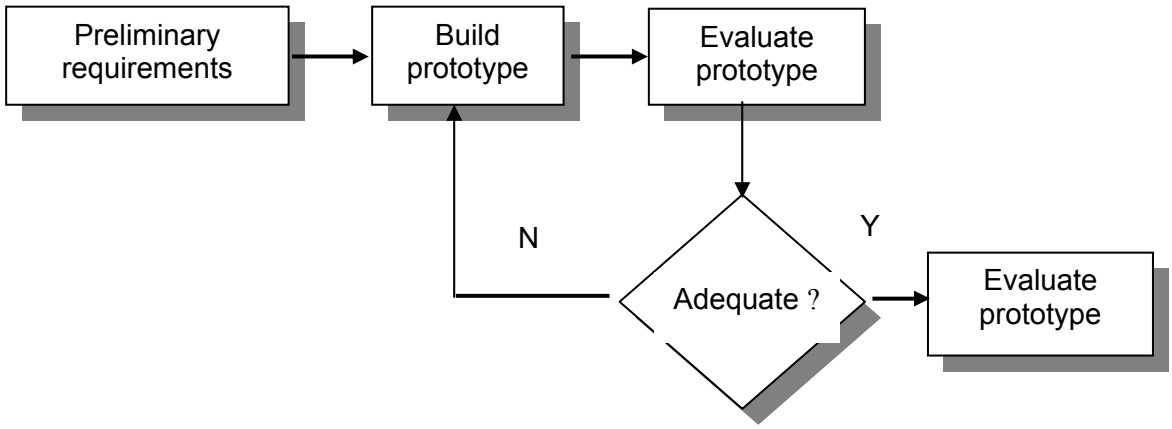
Desain Iteratif dan Prototyping

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.

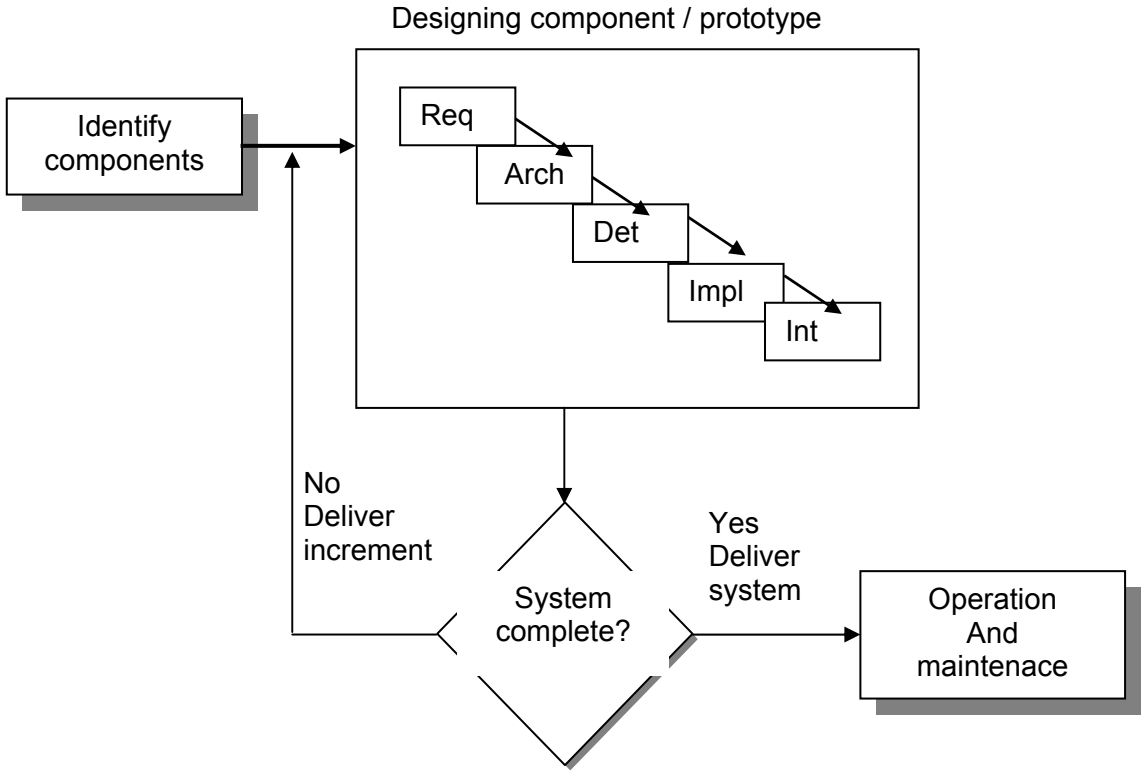
Pada sisi tehnik, desain iteratif dideskripsikan dengan pengunaan prototype. Prototype merupakan alat yang mensimulasikan beberapa (tidak semua) fitur dari sistem yang akan dibuat. Terdapat tiga pendekatan utama prototyping, yaitu :

- ***Throw-away*** : prototype dibuat dan ditest. Pengalaman yang diperoleh dari pembuatan prototype tersebut digunakan untuk membuat produk akhir (final), kemudian prototype tersebut dibuang (tak dipakai)



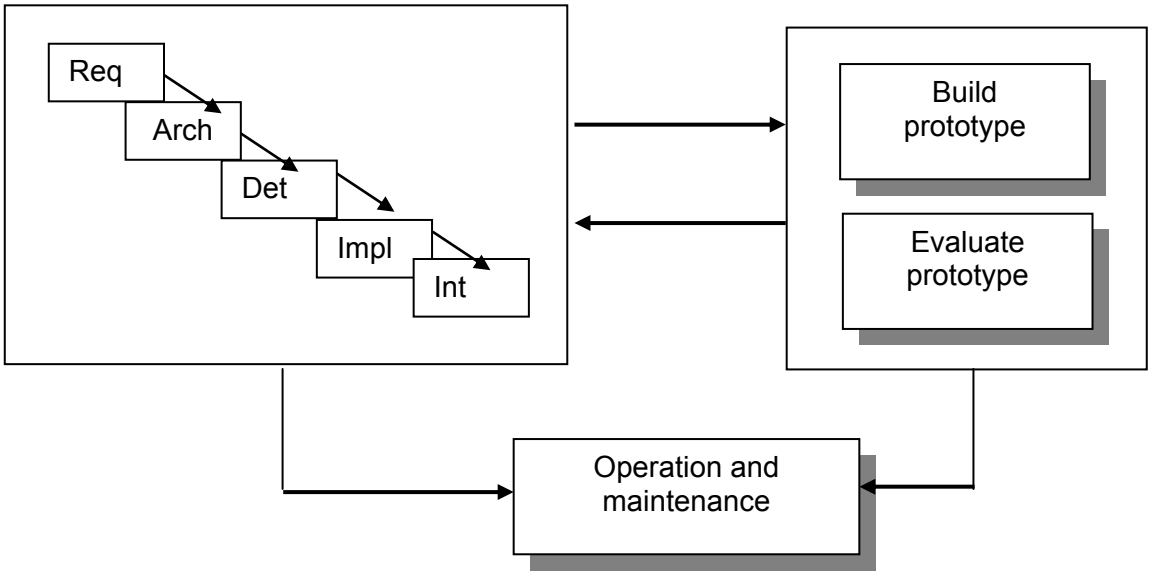
Gambar 3.4 Prototype Model Throw-away

- ***Incremental*** : produk finalnya dibuat sebagai komponen-komponen yang terpisah. Desain produk finalnya secara keseluruhan hanya ada satu, tetapi dibagi-bagi dalam komponen-komponen lebih kecil yang terpisah (independent).



Gambar 3.5 Prototype Model Incremental

- ***Evolutionary*** : Pada metode ini, prototypenya tidak dibuang tetapi digunakan untuk iterasi desain berikutnya. Dalam hal ini, sistem atau produk yang sebenarnya dipandang sebagai evolusi dari versi awal yang sangat terbatas menuju produk final atau produk akhir.



Gambar 3.6 Prototype Model Evolutionary

Disisi manajemen, terdapat beberapa masalah potensial yang terkait dengan prototyping, seperti :

- **Waktu**, membangun prototype membutuhkan waktu, sehingga seringkali prototype dipakai jika waktunya cepat. Hingga muncul istilah rapid prototyping.
- **Rencana**, sebagian manajer proyek tidak memiliki pengalaman untuk menyatukan proses prototyping dengan keseluruhan rencana perancangan.
- **Fitur Non-fungsional,** seringkali fitur sistem yang paling penting merupakan fitur non-fungsional seperti safety dan reliability, tidak disertakan dalam prototyping.
- **Kontrak,** proses desain kadang dibatasi oleh kontrak antara desainer dengan customer yang mempengaruhi aspek tehnik dan manajerial.

Tehnik-tehnik Prototyping

Terdapat beberapa tehnik yang dapat dipergunakan untuk membuat rapid prototype, seperti :

- **Storyboard**, adalah bentuk prototype yang paling sederhana berupa gambaran secara grafis dari tampilan sistem yang akan dibangun tanpa fungsi dari sistem.
- **Simulasi Fungsi Terbatas,** fungsi sistem disertakan pada prototype tidak sekedar gambar tampilannya saja.
- **High-level Programming Support**, HyperTalk adalah contoh dari special-purpose high-level programming language yang memudahkan desainer membuat fitur tertentu dari sebuah sistem interaktif.

Rasionalitas 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*), anggota tim memahami keputusan kritis / penting mana yang telah dibuat, alternatif apa saja yang telah diteliti, dan alasan apa yang menyebabkan suatu alternatif dipilih diantara alternatif lainnya.

- 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.

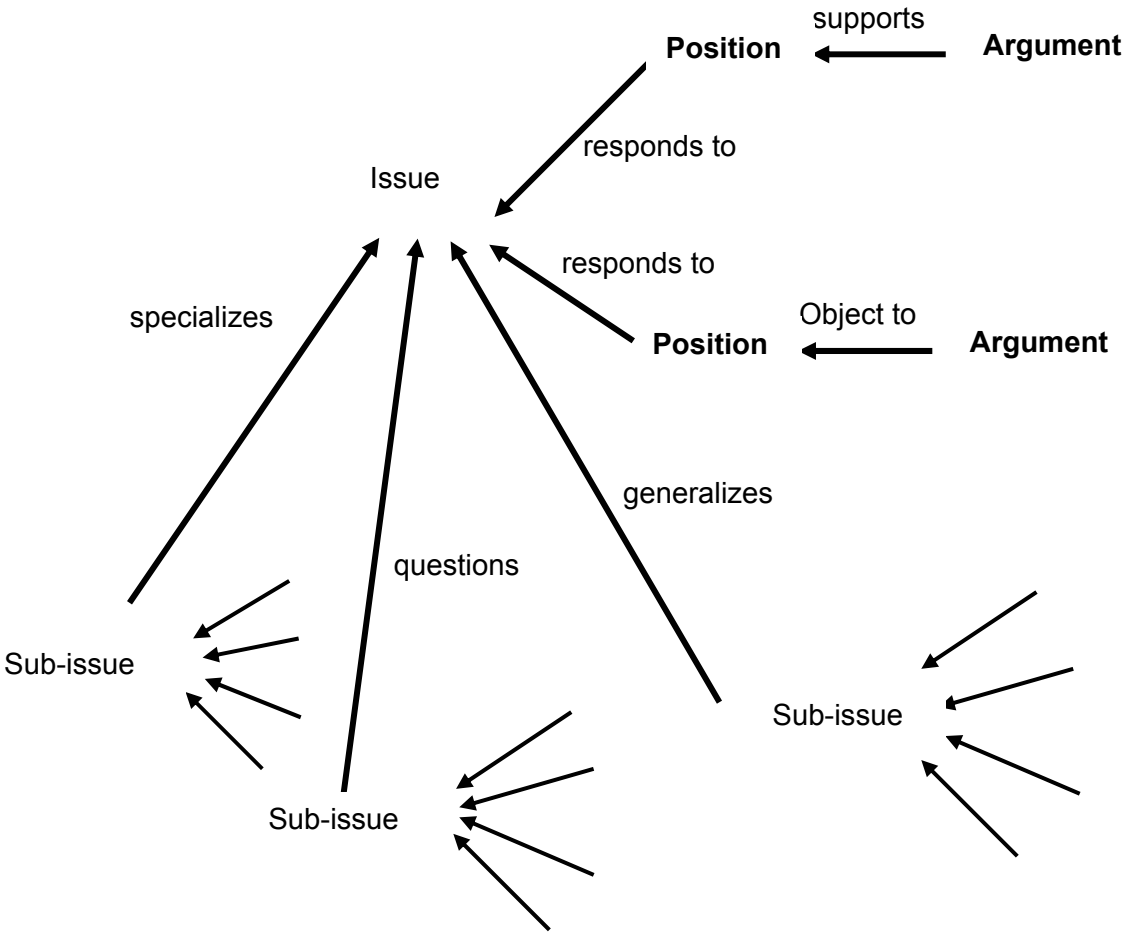
Pada area IMK, rasionalitas desain secara khusus memiliki arti penting untuk beberapa alasan :

- Umumnya tidak ada satu alternatif desain yang terbaik. Desainer dihadapkan pada kondisi trade-off antara alternatif berbeda yang ada. Rasionalitas desain digunakan untuk mendaftar pilihan yang ada dan mengkomunikasikan pilihan tersebut.
- Meskipun terdapat solusi yang optimal, ruang lingkupnya terlalu besar untuk langsung dapat ditemukan. Sehingga perlu desainer perlu mengindikasikan semua alternatif yang telah diselidiki.
- Usability sistem interaktif sangat bergantung pada konteks penggunaannya. Memperhatikan konteks keputusan perancangan yang dibuat akan membantu proses perancangan sistem yang baru nantinya. Jika konteksnya sama dengan yang lama, maka rasionalitas desain dapat diadopsi tanpa revisi, sebaliknya jika konteksnya berubah, maka rasionalitas desain ditelaah kembali dan dihilangkan alternatif yang tidak sesuai.

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

Sebagian besar rasionalitas desain mengadaptasi bentuk ***issue-based information system*** (IBIS) yang merepresentasikan 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***.

Versi grafis dari IBIS kemudian dikembangkan oleh Conklin dan Yakemovic dan disebut sebagai gIBIS, seperti yang ditunjukkan pada gambar 3.7 berikut.



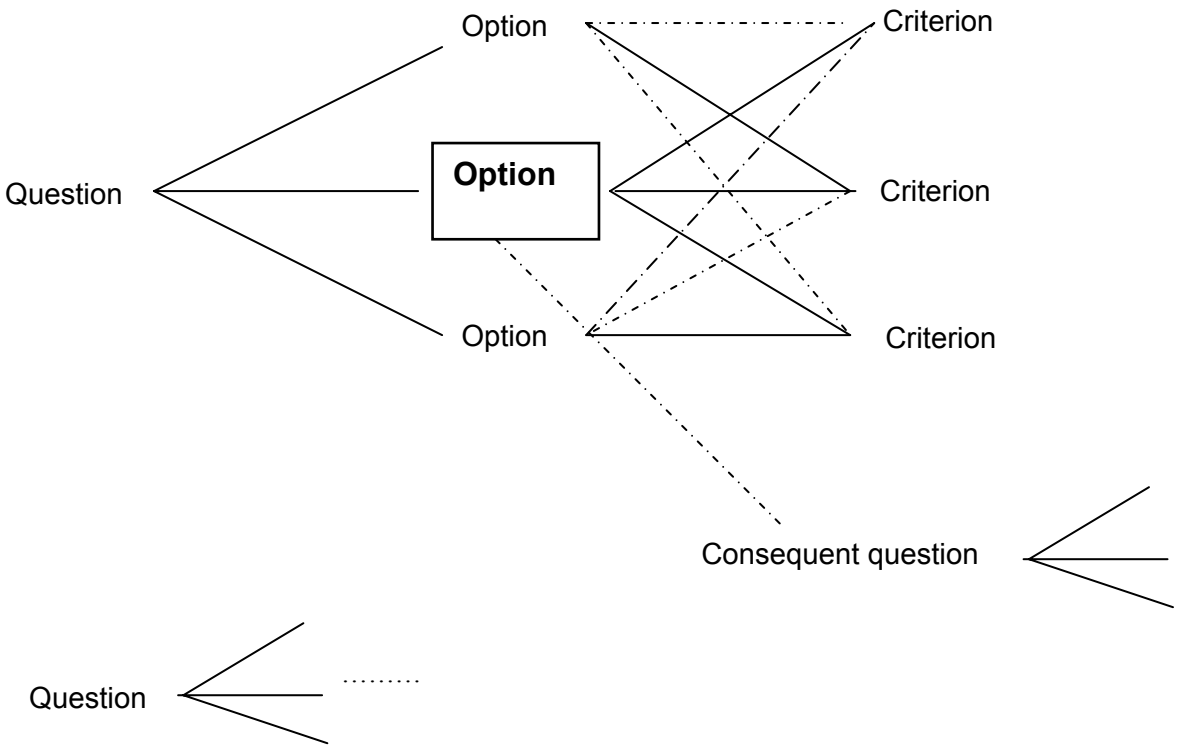
Gambar 3.7 Struktur gIBIS Design Rationale

IBIS dimaksudkan untuk digunakan selama pertemuan atau diskusi yang dilakukan pada saat proses perancangan. IBIS dipakai sebagai alat untuk merekam dan menstrukturkan masalah yang diangkat serta keputusan yang diambil terhadap masalah tersebut.

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*.

Struktur ruang desain ini dimulai dengan ***Question*** yang merupakan masalah utama dalam perancangan. ***Option*** memberikan alternatif solusi terhadap question. Option yang ada dinilai berdasarkan beberapa ***criteria*** untuk menentukan mana yang paling menguntungkan. Struktur *design space analysis* dapat dilihat pada gambar 3.8 berikut.



Gambar 3.8 QOC (Questions, Options, Criteria) Notation

Kunci dari *design space analysis* yang efektik terletak pada pemilihan question yang benar dan criteria yang tepat untuk menilai option. Question awal yang diangkat harus cukup umum sehingga dapat mengakomodasi sebagian besar ruang desain yang mungkin namun juga spesifik sehingga dapat diberikan option yang jelas. Menentukan criteria terhadap suatu option yang tepat dapat menjadi suatu hal yang sulit. Tehnik QOC menyarankan untuk menggunakan criteria yang umum seperti prinsip usability.

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. Sebagai contoh, dalam perancangan program bantuan untuk mengoperasikan Smalltalk, programer akan melakukan tugas yang menjawab pertanyaan-pertanyaan berikut :

- Apa yang dapat saya lakukan ? Operasi atau fungsi apa yang dapat dilakukan pada lingkungan pemrograman ini ?
- Bagaimana program ini bekerja ? Bagaimana fungsi-fungsi yang ada bekerja ?
- Bagaimana saya menggunakannya ? Begitu saya mengetahui operasi yang akan saya lakukan, bagaimana membuat programnya ?

Untuk setiap pertanyaan, dibuat sekumpulan skenario perilaku user dan sistem untuk menjawab pertanyaan 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.

Latihan :

1. Paradigma yang memanfaatkan pengetahuan atau kebiasaan user terhadap suatu hal dan digunakan sebagai elemen interface, misalnya icon adalah :
   A. Metaphor
   B. Computer-supported cooperative work
   C. Programming toolkit
   D. Multi-modality

2. Berikut ini adalah prinsip yang mendukung pendayagunaan, ***kecuali*** :
   A. Learnability
   B. Availability
   C. Flexibility
   D. Robustness

3. Salah satu prinsip yang mempengaruhi robustness, yang berhubungan dengan bagaimana user mengetahui / menyadari laju komunikasi dengan sistem adalah :
   A. Observability
   B. Recoverability
   C. Responsiveness
   D. Task conformance

4. Proses pemeriksaan apakah perangkat lunak yang kita buat dapat berjalan sesuai dengan fungsinya disebut :
   A. Validasi
   B. Verifikasi
   C. Komunikasi
   D. Standarisasi

5. Pendekatan prototyping yang tetap menggunakan prototype sebelumnya untuk iterasi desain berikutnya adalah :
   A. Throw-away
   B. Incremental
   C. Evolutionary
   D. Time-sharing