

BAB XVII

SEVEN FUNDAMENTAL DESIGN PRINCIPLE

Yusri, Riyeen, dan Dorman: Pintu

Siang itu Yusri, Riyeen, dan Dorman tampak sedang makan bareng di kantin kampus. Sambil menunggu pesanan mereka datang, mereka seperti biasa saling bersendau gurau. Biasalah cowok, kadang awalnya ngobrolin apa, terus tiba-tiba belok ke mana. Tiba-tiba saja mereka sampai pada topik yang entah bagaimana bisa sampai ke situ. Entah siapa yang membawanya ke sana.

Dan Dorman tiba-tiba ngomongin soal pintu.

“Kepikiran ga sih, pintu itu kurang ajar loh. Masa pintu aja harus pakai tulisan “Dorong” dan “Tarik”! kayak kita ga ngerti aja cara buka pintu!”

Kata-kata Dorman itu membuat Yusri dan Riyeen diam sejenak. Dan tak lama tawa meledak di antara mereka bertiga. Baik Yusri dan Riyeen tiba-tiba menyadari keanehan mengapa pintu perlu dipasang tulisan-tulisan semacam itu.

“Menghina banget kan yang bikin pintu!”

Perkataan Dorman itu langsung direspon dengan anggukan kepala dari Yusri dan Riyeen. Bahkan Riyeen menambahkan jempol kepada Dorman.

“Tapi ada yang lebih parah. Pintu yang dikasih “Dorong” tapi bisa ditarik atau Pintu yang dikasih “Tarik” tapi bisa didorong. Itu pintu tukang tipu!”

Tambahan dari Riyeen itu membuat Yusri dan Dorman makin keras tertawa.

“Eh pintu di ruang kuliah kita di lantai 2 itu kan juga nyebelin. Kan harusnya didorong yah! Tapi kan tu pintu ada handle-nya. Jadi sering kalau pas aku mau terlambat malah aku tarik tuh pintu.”

Riyeen dan Dorman juga cerita kalau mereka pernah punya pengalaman yang sama dengan Yusri. Kalau saat biasa sih ga ada masalah. Bahkan kadang-kadang pintu tersebut dibiarkan terbuka. Tapi kalau pas telat dan pintunya tertutup, mereka ga sengaja menarik pintu yang harusnya didorong. Pasti habis itu bikin dosen dan mahasiswa yang di dalam ruangan pada ketawa.

Kemudian mereka bertiga menambah beberapa contoh cerita lucu tentang pintu. Entah siapa yang memulai, tiba-tiba obrolan berganti menjadi jendela. Kemudian entah siapa tiba-tiba jadi ngomongin gadget.

Namanya juga orang ngobrol.

2 Jurang Pemisah User dan Sistem

Semua orang, termasuk kalian, pasti pernah mengalami momen pertama menggunakan sebuah produk atau aplikasi/sistem interaktif.

Apakah kalian ingat kapan pertama kali menggunakan Microsoft Word?

Kapan pertama kali menggunakan web belanja online?

Kapan pertama kali menggunakan sosmed?

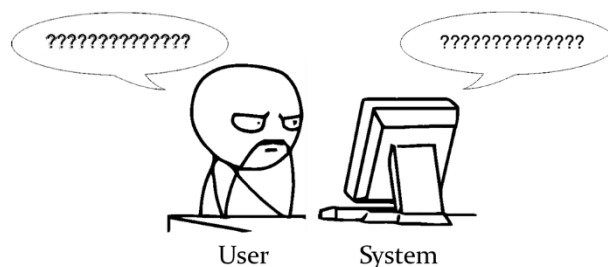
Kapan pertama kali menggunakan software desain grafis?

Mungkin ada yang cepat menguasai. Mungkin ada yang butuh waktu. Mungkin ada yang sampai sekarang tidak bisa pakai.

Apapun itu, pasti ada momen “bingung” atau “tidak tahu” yang pernah kalian rasakan saat pertama kali menggunakan produk.

Itu hal biasa dan manusiawi.

Setiap user pasti pernah mengalami kebingungan saat menggunakan aplikasi/sistem interaktif (Gambar 1). Kebingungan ini paling terasa saat di awal. Tapi tentu tidak menutup kemungkinan kebingungan ini terjadi setelah melalui masa-masa awal tadi.



Gambar 1. User dan System sama-sama bingung

Tapi, jika aplikasi/sistem interaktif bisa ngomong, mungkin mereka juga akan menyampaikan kebingungan.

“ini yang makai tahu ga sih cara makai yang bener”

“kok kayak gitu makainya”

“kok dia marah-marah sama aku?”

Tentunya agar interaksi yang harmonis terjadi antara user dengan aplikasi/sistem interaktif, maka kebingungan-kebingungan ini harus diminimalisir.

Tapi, mengapa kebingungan ini terjadi?

Dan bagaimana cara meminimalisir nya?

Mari coba telaah lebih dalam dan lebih perlahan jawaban dari dua pertanyaan tersebut.

Mari kita lihat sebuah remote AC



Saat melihatnya kalian akan sadar ada banyak tombol yang bisa ditekan. Di setiap tombol ada teks dan icon yang menyertainya.

Kalian mungkin akan berpikir

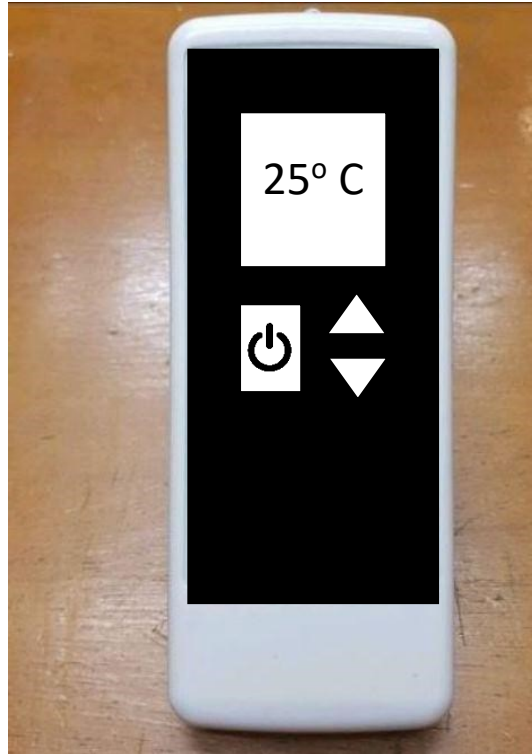
“Dengan remote ini saya bisa ngapain aja?”

“ini ada tombol buat apa aja?”

Atau yang paling sederhana

“AC itu punya banyak fungsi ya? Tak kira cuma lebih panas atau dingin”

Mungkin kita berharap remote AC itu bentuknya



Kalian tidak tahu apa saja yang bisa kalian lakukan dengan remote itu.

Mungkin kalian tahu kalau tombol-tombol itu bisa ditekan.

Tapi kalian tidak tahu tombol apa buat apa.

Kalian tidak tahu apa maksud teks dan icon yang menemani setiap tombol.

Tapi okelah, kalian mencoba bertualang dengan menekan sebuah tombol.

Setelah menekan, kalian melihat reaksi yang terjadi pada AC.

Kalian merasa setelah menekan sebuah tombol, ada perubahan yang terjadi pada AC. Dari situ kalian memahami apa efek tombol itu pada AC.

Tapi ada tombol-tombol yang sepertinya tidak membuat perubahan.

“apa nekannya kurang kuat?”

“tapi tadi AC-nya bunyi habis tombol ditekan”

“Tapi kok kayaknya ga ada yang berubah?”

Kalian coba perhatikan, coba dengarkan, coba rasakan. Tapi kalian tetap tidak bisa merasakan perubahan apapun dari AC tersebut.

Akhirnya, sampai sekarang kalian hanya menggunakan tombol-tombol yang kalian tahu. Dan itupun mungkin hanya sedikit dari semua tombol yang disediakan.

Mungkin orang cerdas di antara kalian akan berkata

“Kenapa tidak baca manual-nya saja?!”

Tapi orang-orang normal akan menjawab

“Manual?”

Ya, tidak semua orang mau membaca manual. Itu juga manusiawi.

Ok kita akhiri cerita kita, dan mari kita jawab satu dari dua pertanyaan tadi

Mengapa kebingungan ini terjadi?

Dari cerita tadi, ada dua alasan mengapa kebingungan terjadi

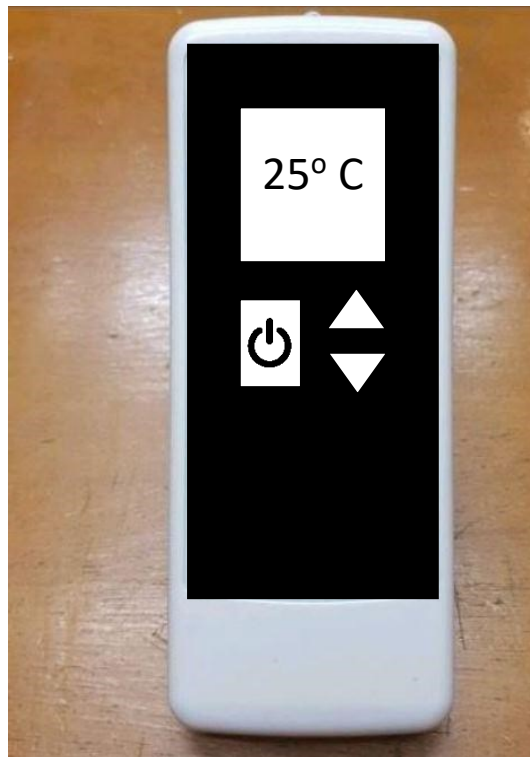
Pertama, user tidak tahu apa yang bisa dilakukan.

Di sini muncul gap antara apa yang diinginkan atau dipikirkan user dengan apa yang diberikan atau diperbolehkan oleh aplikasi/system interaktif.

AC hanya memberikan remote untuk mengontrolnya, tapi user tidak tahu apa yang bisa dilakukan dengan remote tersebut.

Di sinilah muncul gap.

Jika remote-nya seperti ini



Mungkin gap nya akan lebih kecil karena user bisa membayangkan apa yang bisa ia lakukan dengan remote tersebut.

Kondisi dimana muncul gap antara apa yang diinginkan atau dipikirkan user dengan apa yang diberikan atau diperbolehkan oleh aplikasi/system interaktif ini disebut dengan **Gulf of Execution** (Jurang Eksekusi).

Gulf of Execution membuat user tidak tahu apa yang bisa ia lakukan.

Kedua, User tidak tahu apa yang terjadi.

Di sini muncul gap antara ekspektasi user dengan reaksi atau output yang diberikan oleh aplikasi/system interaktif.

Ketika sebuah tombol dicoba ditekan oleh user, tentunya ia berharap ada reaksi yang diberikan AC.

Reaksi inilah yang membuat user memahami apa fungsi tombol itu.

Jadi walaupun awalnya ada **Gulf of Execution**, gap ini pelan-pelan sirna karena user bisa menangkap apa yang terjadi pada AC dan menghubungkannya dengan tombol tersebut.

Tapi ketika tidak ada reaksi, user akan bingung lagi.

Sebenarnya bukan tidak ada reaksi.

Kan ga mungkin ada tombol yang sudah capek-capek dibuat, ternyata ketika ditekan tidak ada reaksi sama sekali.

Tapi yang terjadi sebenarnya, AC itu memberikan reaksi. AC itu memberikan output.

Tapi output itu tidak ditangkap oleh user. Output itu dalam bentuk yang tidak sesuai ekspektasi user.

Mungkin output nya butuh waktu agar terasa.

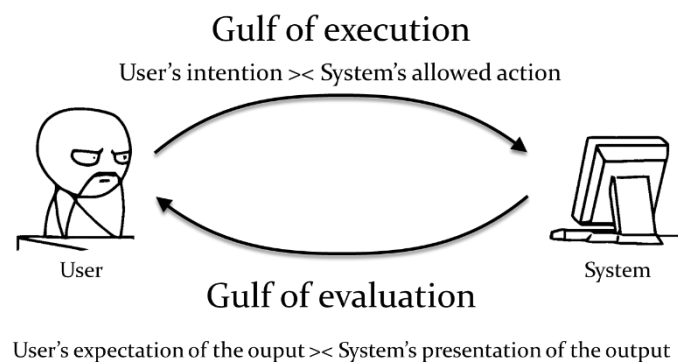
Mungkin outputnya tidak melulu berkaitan dengan suhu. Misal timer on/off atau gerakan fan.

Apapun itu, output yang diberikan AC tidak sesuai ekspektasi user sehingga ia bingung.

Kondisi dimana muncul gap antara ekspektasi user dengan reaksi atau output yang diberikan oleh aplikasi/system interaktif ini disebut dengan **Gulf of Evaluation** (Jurang Evaluasi).

Gulf of Evaluation membuat user tidak tahu apa yang terjadi

Sehingga jika digambarkan sederhana, kebingungan itu dapat dilihat pada Gambar 2.



Gambar 2. Jurang Pemisah User dan Aplikasi/Sistem Interaktif

Inilah 2 kebingungan yang selalu kita rasakan saat menggunakan produk atau aplikasi/sistem interaktif apapun.

Menghilangkan kebingungan berarti melakukan usaha untuk mempersempit dua jurang yang muncul. Tidak bisa hanya salah satunya saja.

Good Design by Donald Norman

Konsep **Gulf of Execution** dan **Gulf of Evaluation** dikemukakan oleh Donald Norman (Gambar 3)



Gambar 3. Donald Norman

Beliau bisa dikatakan salah satu founding fathers dari HCI. Penemu istilah UX. Profesor di bidang Cognitive Psychology. Idola saya di dunia UX Design.

Baik **Gulf of Execution** maupun **Gulf of Evaluation** sebenarnya merupakan keniscayaan yang manusiawi. Donald Norman memaklumi hal tersebut.

Salah satu quote beliau (sudah saya terjemahkan)

“ Tidak masalah jika anda salah di percobaan pertama.

Tidak masalah jika anda harus membaca buku atau minta diajarkan seseorang.

Tapi akan jadi masalah jika anda harus melakukan hal yang sama untuk kedua kalinya “

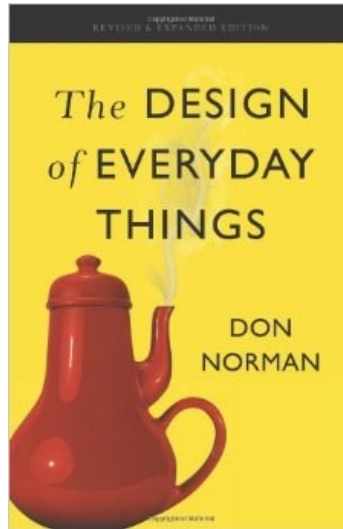
Jadi kebingungan itu manusiawi. Yang tidak manusiawi adalah bingung terus menerus tanpa adanya progress atau kemajuan.

Makanya Donald Norman menekankan pentingnya **GOOD DESIGN** untuk memperkecil **Gulf of Execution** dan **Gulf of Evaluation**. Untuk membuat user merasa ada kemajuan yang ia rasakan.

Jika user tidak pernah merasakan kemajuan selama menggunakan aplikasi/sistem interaktif, maka dijamin lama kelamaan ia akan meninggalkan aplikasi/sistem interaktif tersebut.

Menurut Norman, **GOOD DESIGN** dibutuhkan di semua produk, sebab **Gulf of Execution** dan **Gulf of Evaluation** bisa terjadi dimana saja.

Salah satu hal yang banyak dibahas di buku beliau, “The Design of Everyday Things”, percaya atau tidak, adalah pintu.



Di awal buku beliau membahas betapa pintu, sebuah produk yang harusnya sederhana, ternyata masih perlu menuliskan manual cara menggunakannya.

Sebuah pintu, ternyata masih membuat orang tertipu apakah ditarik, didorong, atau digeser.

Jika yang tertipu hanya satu orang, salah orangnya. Tapi jika banyak, maka ya salah pintunya

Itulah sebabnya kemudian muncul istilah “Norman Door”

Disclaimer: Google aja!!! kalian akan belajar banyak

Jadi gurauan absurd Yusri, Riye, dan Dorman itu menjadi salah satu contoh sederhana bahwa **GOOD DESIGN** dibutuhkan dimana-mana.

Mari kita masuk ke pertanyaan kedua di bagian sebelumnya

Bagaimana cara meminimalisir kebingungan tersebut?

Jawaban sederhananya adalah meminimalisir **Gulf of Execution** dan **Gulf of Evaluation**.

Tapi jawaban ini tidak membantu.

Menurut Norman, kunci dari **GOOD DESIGN**, kunci dari meminimalisir **Gulf of Execution** dan **Gulf of Evaluation**, sebenarnya terletak pada dua hal.

DISCOVERABILITY + UNDERSTANDING

DISCOVERABILITY maksudnya user bisa menemukan apa saja yang bisa ia lakukan, dimana ia bisa melakukannya, dan bagaimana melakukannya. **DISCOVERABILITY** juga berarti bahwa user bisa menemukan atau merasakan reaksi dari suatu tindakan.

Bayangkan jika ada remote ac yang tombol-tombolnya ada yang disembunyikan.

Ga usah jauh-jauh ngomongin cara makainya, kalau tombolnya aja disembunyikan, apa user bakal bisa menggunakannya dengan benar?

Atau ingat lagi contoh tadi, habis tombol ditekan, kita tidak merasakan ada perubahan

Kalau user aja ga bisa merasakan ada perubahan, apa user bakal bisa menggunakannya dengan benar?

Maka kunci awal meminimalisir **Gulf of Execution** dan **Gulf of Evaluation** adalah memastikan user bisa menemukan aksi apa saja yang bisa ia lakukan, dimana melakukannya, dan bagaimana melakukannya. Dengan kata lain, meningkatkan **DISCOVERABILITY**.

Kalau user sudah menemukan dia bisa apa saja, langkah selanjutnya adalah **UNDERSTANDING**, yaitu membuat mereka paham apa yang mereka lakukan atau membuat paham apa yang sedang terjadi.

Walaupun kita menemukan tombol, kita bisa scroll, kita bisa mengetik di textbox, kita bisa memilih dari list, tapi jika tidak paham apa artinya...ya percuma saja.

Menemukan sesuatu tidak berarti kita memahami sesuatu.

Itulah sebabnya untuk lebih baik lagi dalam meminimalisir **Gulf of Execution** dan **Gulf of Evaluation**, maka kita perlu memastikan user paham apa yang mereka lakukan atau paham apa yang sedang terjadi. Dengan kata lain, meningkatkan **UNDERSTANDING**.

Bagaimana cara meningkatkan **DISCOVERABILITY** dan **UNDERSTANDING**?

Norman kemudian menyusun **SEVEN FUNDAMENTAL DESIGN PRINCIPLE** untuk memberi guideline bagaimana caranya meningkatkan **DISCOVERABILITY** dan **UNDERSTANDING**.

SEVEN FUNDAMENTAL DESIGN PRINCIPLE tersusun atas:

- **CONCEPTUAL MODEL**
- **DISCOVERABILITY**
- **AFFORDANCE**
- **SIGNIFIER**
- **CONSTRAINT**
- **FEEDBACK**
- **MAPPING**

SEVEN FUNDAMENTAL DESIGN PRINCIPLE merupakan contoh sebuah Design Principle. Bab 17 ini akan saya gunakan untuk membahas hal tersebut.

Mengapa saya memilih mengajarkan **SEVEN FUNDAMENTAL DESIGN PRINCIPLE**?

Karena saya menyukai filosofi desain dari Donald Norman

Beliaulah yang menyadarkan saya pentingnya membuat aplikasi dengan fokus pada user.

Selain itu, konsep **SEVEN FUNDAMENTAL DESIGN PRINCIPLE** bisa digunakan untuk produk apapun, tidak harus aplikasi/sistem interaktif. Makanya saya suka!!!

Tapi ini bukan paksaan bahwa dalam kuliah dan project kalian harus pakai **SEVEN FUNDAMENTAL DESIGN PRINCIPLE**.

Banyak UX Design Pattern di luar sana. Silahkan pelajari dan gunakan yang sesuai dengan kebutuhan kalian!!!

Conceptual Model

Ketika pertama kali menggunakan aplikasi/sistem interaktif, jelas user buta tentang aplikasi/sistem interaktif tersebut.

Namun semakin lama mencoba, semakin bertambah pengetahuan user tentang aplikasi/sistem interaktif tersebut.

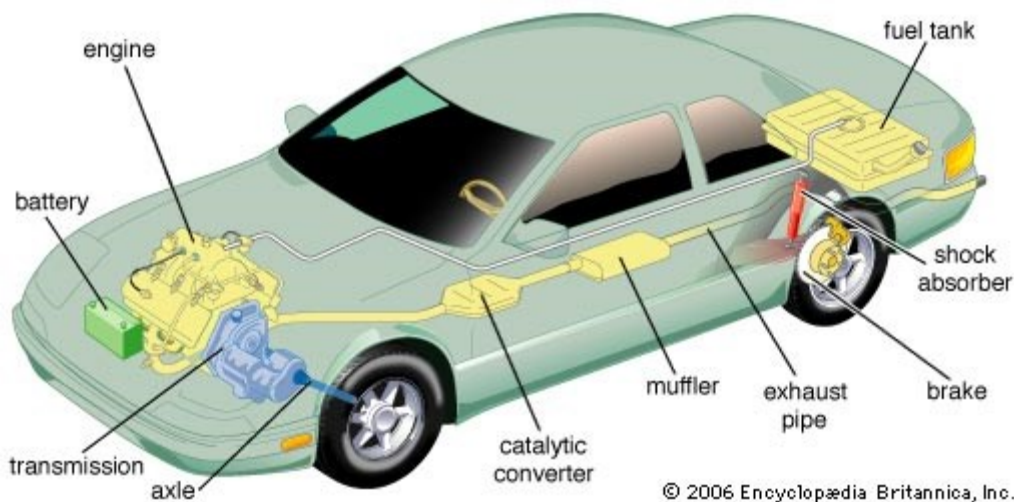
Di dalam pikirannya, pelan-pelan akan terbayang sebuah model yang menggambarkan atau memperkirakan cara kerja aplikasi/sistem interaktif tersebut.

Model inilah yang dikenal dengan nama **Mental Model**, yaitu...

Model dalam pikiran seseorang yang merepresentasikan pemahaman mereka tentang bagaimana sebuah produk bekerja/berfungsi

Mental Model biasanya sederhana dan jauh dari cara kerja sistem sebenarnya.

Misalnya mobil, cara kerja mobil itu seharusnya



Tapi bagi kalian yang bisa mengendarai mobil, **Mental Model** kalian utamanya akan berisi:

- Gas → bergerak
- Rem → melambat
- Kopling + Perseneling → ganti gigi
- Setir → belok

As simple as that

Memang sederhana....tapi cukup tidak untuk menyetir mobil?

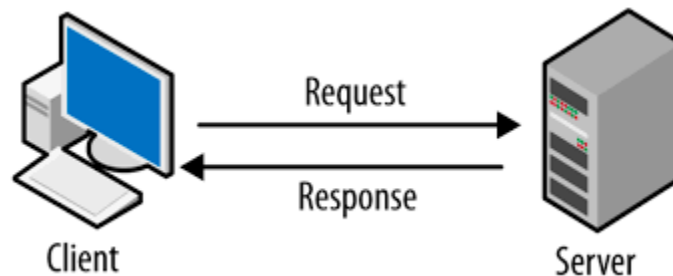
Kenapa cukup?

Karena **Mental Model** mobil tersebut merupakan bentuk sederhana cara kerja mobil sebenarnya.

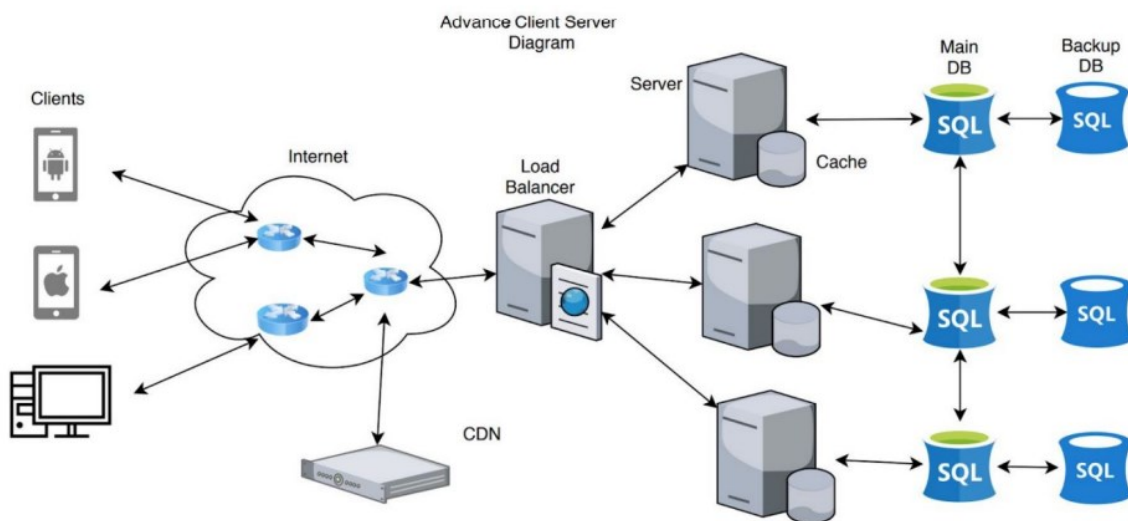
Artinya...walaupun sederhana, tapi dia tidak jauh dari kenyataan.

Atau misal saat menggunakan website atau web app.

Mungkin pikiran sederhana kita hanya...



Padahal mungkin aslinya...



Tapi cukup tidak untuk bisa menggunakan website atau web app?

Jadi memiliki **Mental Model** sederhana itu wajar dan tidak masalah. Asalkan, **Mental Model** yang dibuat menyerupai cara kerja aplikasi/sistem interaktif yang sebenarnya.

Namun jika **Mental Model** tidak menyerupai cara kerja aplikasi/sistem interaktif yang sebenarnya, maka di situ terjadi hambatan untuk **UNDERSTANDING**.

Tapi ingat!!!

Mental Model akan berkembang juga seiring user berinteraksi dengan aplikasi/sistem interaktif tersebut.

Jadi jangan panik ketika **Mental Model** user salah. Karena masih ada momen untuk dibenarkan.

Itu seperti kalian mengira sebuah pintu untuk didorong ternyata harusnya ditarik. Pintu yang sempurna tentunya tidak akan membuat kita salah.

Tapi mencapai sempurna jangan sampai membuat kita loyo atau panik ketika user melakukan kesalahan.

Yang paling penting adalah bukan mencapai sempurna dalam sekali percobaan. Melainkan bergerak mendekati kesempurnaan itu.

Makanya Norman mengatakan tidak masalah ketika user salah, asalkan ada kemajuan di percobaan-percobaan berikutnya.

Seperti yang saya katakan sebelumnya, kebingungan itu manusiawi. Yang tidak manusiawi adalah bingung terus menerus tanpa adanya progress atau kemajuan.

Oleh karena itu, tidak perlu panik ketika **Mental Model** di awal salah, asalkan pelan-pelan dia bisa berkembang dan mengarah ke yang seharusnya.

Kita baru panik kalau setelah beberapa kali menggunakan, ternyata **Mental Model** user tidak kunjung benar.

Bagaimana kemudian mengatasi ini?

Norman kemudian mengenalkan **CONCEPTUAL MODEL** sebagai salah satu **SEVEN FUNDAMENTAL DESIGN PRINCIPLE**.

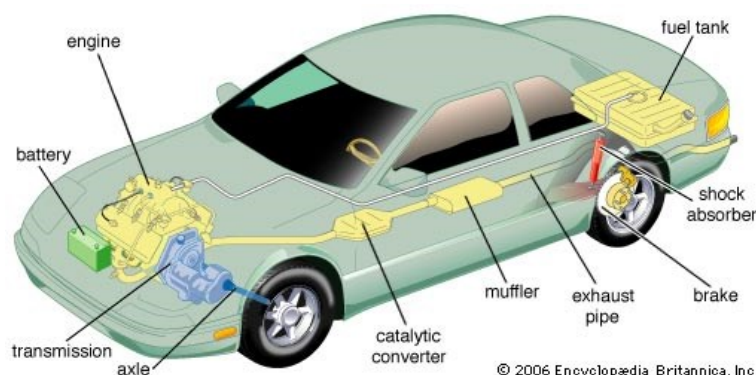
*Disclaimer: Namanya memang sama dengan yang digunakan di bab 15, tapi khusus untuk bab 17 ini, istilah **CONCEPTUAL MODEL** saya gunakan untuk mengacu pada principle dari Norman*

Menurut Norman, pembentukan **Mental Model** tidak bisa dibiarkan menjadi perjuangan user saja. Tetapi UX Designer perlu untuk membantu pembentukan **Mental Model** tersebut.

Caranya, kita perlu bijak dan cerdas dalam menunjukkan bagaimana cara kerja aplikasi/sistem interaktif kita ke user.

Atau dalam konsep Norman, kita perlu bijak dan cerdas dalam menampilkan **CONCEPTUAL MODEL** dari aplikasi/sistem interaktif.

Misal mobil tadi



© 2006 Encyclopædia Britannica, Inc.

Memang itu adalah **CONCEPTUAL MODEL** mobil yang sebenarnya. Jika mobil itu dibedah dan diperlihatkan semua seisinya, kita pengguna mobil mungkin akan terkejut.

Tapi kenapa mobil bisa digunakan walaupun **CONCEPTUAL MODEL** rumit?

Karena bagian yang ditampilkan ke user adalah bagian yang sederhana. Dalam artian, UI nya sederhana

UI nya ya singkatnya cuma:

- Gas
- Rem
- Kopling
- Perseneling
- Setir

Cukup dengan 5 UI element tersebut kita sudah bisa menyetir.

Jadi betapa hebatnya UX Designer mobil, **CONCEPTUAL MODEL** yang kompleks bisa diubah dan ditampilkan menjadi **CONCEPTUAL MODEL** yang sederhana.

Bahkan mobil matic membuatnya lebih sederhana lagi dengan menghilangkan kopling.

Atau contoh di aplikasi/sistem interaktif, misalnya aplikasi untuk tambah data ke database.

CONCEPTUAL MODEL nya harusnya kira-kira:

1. Input alamat dan nama database
2. Input password database
3. Input query
4. Terima return value
5. Tampilkan return value

Tapi lihat bagaimana **CONCEPTUAL MODEL** itu ditampilkan ke user.

CONCEPTUAL MODEL-nya berubah menjadi:

1. Input data yang ingin disimpan
2. Klik submit
3. Lihat hasil

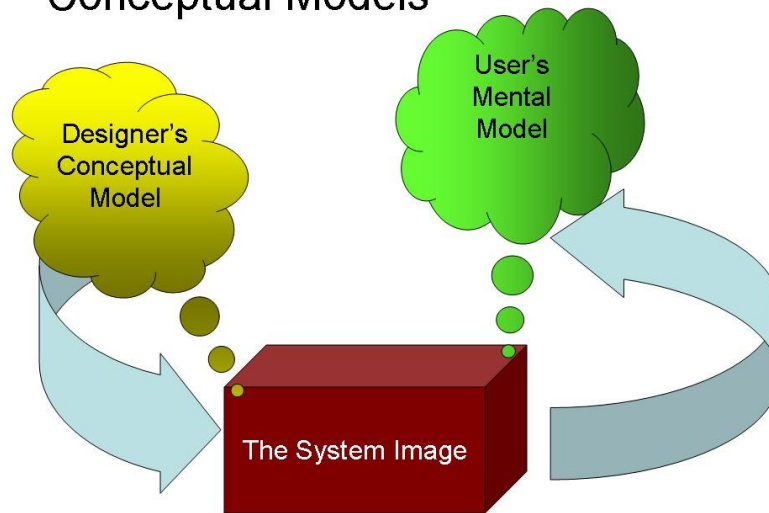
Bisa jadi sederhana kan?

Jadi jika **CONCEPTUAL MODEL** yang ditampilkan sederhana, maka **Mental Model** akan lebih cepat berkembang menuju benar.

Tapi jika **CONCEPTUAL MODEL** yang ditampilkan rumit, maka **Mental Model** juga akan lebih lama berkembang menuju benar.

Di sinilah Norman mengenalkan istilah **System Image**, yaitu bentuk penyederhanaan **CONCEPTUAL MODEL** yang nantinya ditampilkan kepada User (Gambar 4).

Conceptual Models

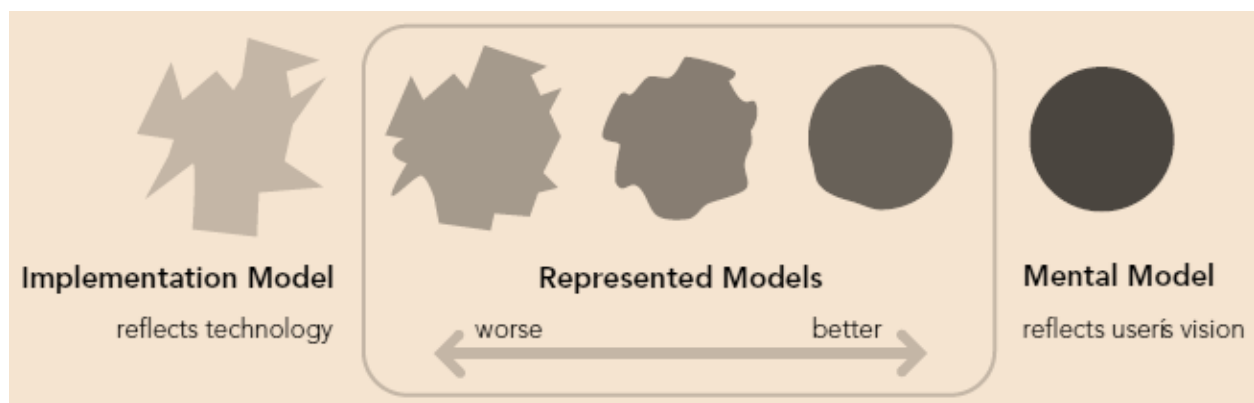


Gambar 4. Hubungan Conceptual Model, System Image, dan Mental Model

Analogikan **CONCEPTUAL MODEL** sebagai bagian dalam mobil, **System Image** sebagai gas, rem, setir, dsb, dan **Mental Model** user adalah apa yang ada di pikiran user tentang mobil.

Jadi **CONCEPTUAL MODEL** boleh rumit, tapi **System Image** harus dicoba untuk dibuat sederhana. Karena bagaimanapun, **Mental Model** user akan terbentuk dari melihat dan berinteraksi dengan **System Image**-nya, bukan **CONCEPTUAL MODEL**.

Semakin sederhana **System Image**, semakin mudah **Mental Model** terbentuk (Gambar 5). Sebaliknya, semakin rumit **System Image**, semakin lama **Mental Model** terbentuk.



Gambar 5. System Image yang baik mendekati Mental Model

Inilah cara kita sebagai UX Designer membantu user berjuang membentuk **Mental Model**-nya.

Dan inilah yang dinamakan principle **CONCEPTUAL MODEL** dari Norman.

Discoverability

Seperti sudah disampaikan sebelumnya, **DISCOVERABILITY** berarti user harus mudah menemukan apa saja yang bisa ia lakukan, dimana ia bisa melakukannya, dan bagaimana melakukannya.

DISCOVERABILITY hanya fokus pada kemudahan untuk menemukan aksi atau output. Sedangkan memahami aksi atau output kita serahkan pada **UNDERSTANDING**.

Jadi kalau kita bicara tombol, maka **DISCOVERABILITY** berarti mudah menemukan tombol itu. Apa fungsi tombol itu, bukan bagian dari **DISCOVERABILITY**.

Kalau kita bicara pesan error, maka **DISCOVERABILITY** berarti mudah menemukan pesan error. Apa fungsi dan cara kerja pesan error itu, bukan bagian dari **DISCOVERABILITY**.

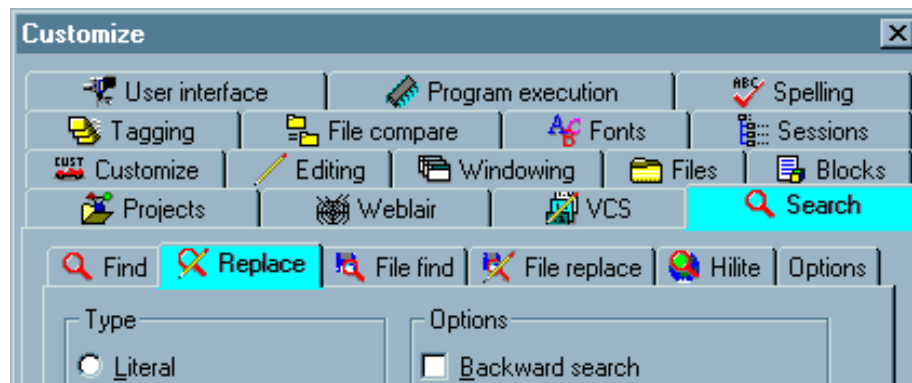
DISCOVERABILITY = Easy to discover

Jadi jika dalam aplikasi/sistem interaktif ada fitur atau output yang sulit ditemukan, maka bisa dikatakan **DISCOVERABILITY**-nya kurang bagus.

Tapi kita perlu bijak dalam menyikapi istilah mudah menemukan ini.

Jangan sampai kalian berpikir bahwa “berarti semua aksi atau semua fitur harus mudah ditemukan?”

Kalau kalian artikan dengan tidak bijak, kalian bisa saja bikin aplikasi/sistem interaktif seperti ini...



Malah nyebelin dan jadinya tidak Usable kan?

Jadi mudah menemukan bukan berarti semua harus mudah ditemukan.

Mudah menemukan bukan berarti semua harus ditampilkan sejak halaman awal.

Jadi bagaimana agar **DISCOVERABILITY** digunakan dengan bijak?

Kuncinya adalah...

Prioritaskan bahwa yang **PALING PENTING** berarti harus yang **PALING MUDAH DITEMUKAN**

Dalam artian

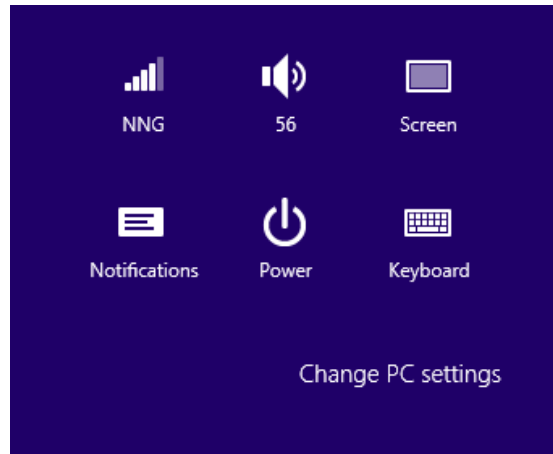
Fitur paling dasar/paling penting harus mudah ditemukan

Output paling dasar/paling penting harus mudah ditemukan

Untuk fitur atau output yang lebih advanced atau yang lebih tidak umum, tidak apa-apa kalau perlu agak berusaha mencarinya.

Contoh aplikasi/sistem interaktif dengan **DISCOVERABILITY** buruk?

Pertama, dari pengalaman saya pribadi, adalah tombol Power di Windows 8

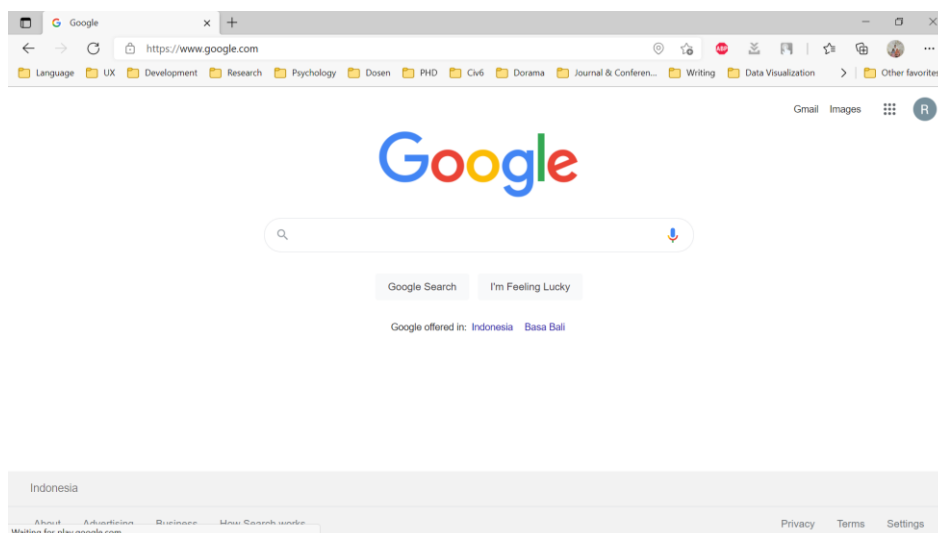


Tombol Power menurut saya merupakan salah satu fitur dasar. Tetapi menemukannya bukan hal yang mudah buat saya karena entah kenapa ada “orang jenius” yang memindahkannya dari kiri bawah layar ke kanan bawah layar.

Contoh aplikasi/sistem interaktif dengan Discoverability baik?

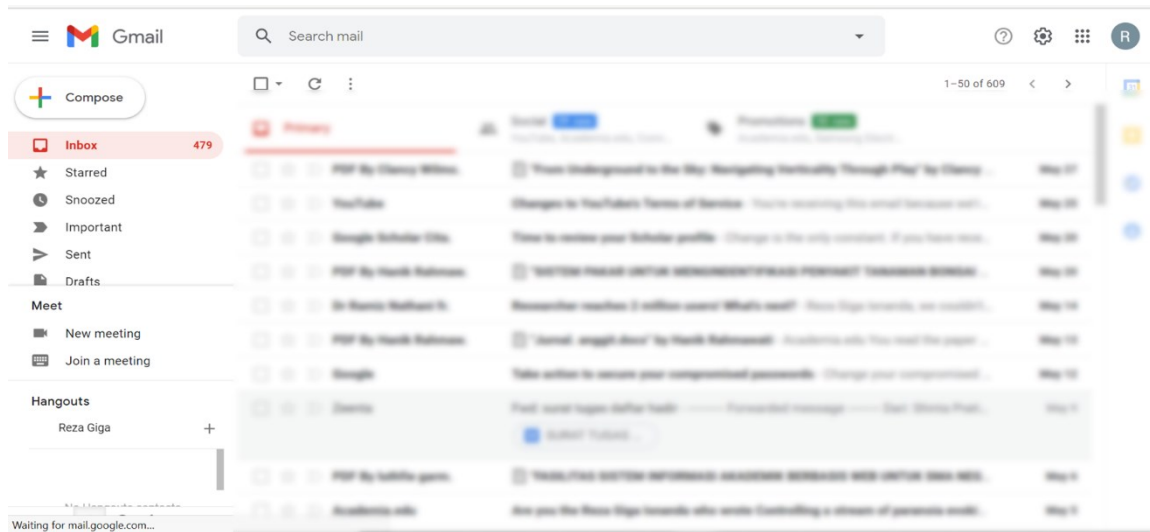
Google

Lihat saja



Sangat mudah kan menemukannya?

Atau Gmail



Fitur dasar dari Gmail jelas adalah baca email dan tulis email.

Dan Gmail jelas memudahkan kita mencari kedua fitur ini.

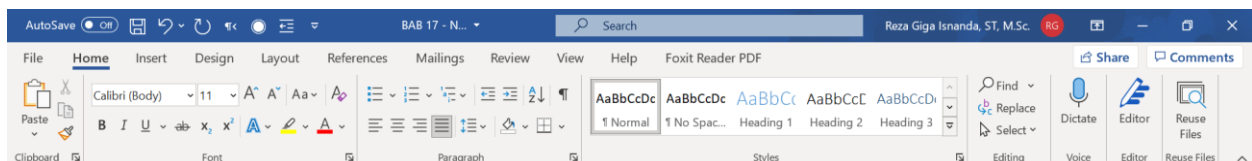
Di sisi kanan jelas terpampang semua email kita.

Tidak hanya itu. Email terbaru selalu diletakkan paling atas, sehingga kita tidak repot mencarinya

Di sisi kiri atas jelas terpampang tombol untuk tulis email.

Apalagi letaknya menyesuaikan cara baca orang Indonesia yang umumnya bergerak dari kiri atas ke kanan bawah. Jadi seolah-olah berkata bahwa fitur email merupakan fitur yang paling awal ditemukan (kiri atas) dari Gmail.

Atau contoh lain adalah Word



Walau fitur Word ada banyak, tapi fitur dasarnya ada di kategori khusus namanya “Home”.

Dan ketika pertama kali membuka Word, Ribbon tab di atas Word selalu berada di posisi “Home”. Bukan “Insert” atau lainnya.

Dan masih banyak contoh lagi.

Jadi walau **DISCOVERABILITY** berarti mudah menemukan, tetap kita harus bijak dalam menggunakannya dalam desain.

Pastikan hal yang paling penting atau paling dasar, baik fitur maupun output, harusnya menjadi hal yang paling mudah ditemukan

Affordance

Dalam bukunya, Norman mengenalkan konsep **AFFORDANCE**.

AFFORDANCE adalah ...

*Relasi antara benda/object fisik dengan manusia/makhluk hidup
yang membolehkan mereka melakukan suatu aksi menggunakan/terhadap object tersebut*

Atau bahasa sederhananya ...

Interaksi yang mungkin terjadi antara seseorang dengan suatu object

Sebagai contoh, antara saya dengan kursi, **AFFORDANCE** yang mungkin terjadi:

- Saya bisa duduk
- Saya bisa mengangkat kursi
- Saya bisa melempar kursi
- Saya bisa berdiri di atas kursi
- Saya bisa men-smackdown orang dengan kursi
- Dan masih banyak lagi

Itulah **AFFORDANCE** yang mungkin terjadi antara saya dengan kursi.

Atau antara saya dengan gelas, **AFFORDANCE** yang mungkin terjadi:

- Saya bisa minum dari gelas
- Saya bisa mengangkat gelas
- Saya bisa melempar gelas
- Saya bisa gunakan gelas sebagai tempat pensil
- Saya bisa gunakan gelas untuk nahan kertas
- Dan masih banyak lagi

Bagaimana dengan aplikasi?

Misal saya dengan Facebook:

- Saya bisa cari teman
- Saya bisa tambah teman
- Saya bisa kirim pesan
- Saya bisa main game
- Saya bisa posting status
- Saya bisa upload foto
- Dan masih banyak lagi

Mengapa kita punya pikiran untuk menggunakan dengan cara itu?

Mengapa kita berpikiran kursi untuk diduduki?

Mengapa kita berpikiran gelas untuk minum?

Mengapa kita berpikiran kirim pesan dengan Facebook?

Karena ada atribut/property dari object tersebut yang kita tangkap.

Misal kursi.

Lihat bentuk kursi.



Bagian warna merah pada kursi merupakan atribut/property dari kursi yang ketika melihatnya, kita menangkap bahwa dia bisa digunakan untuk menaruh sesuatu.

Misalnya.....pantat kita?

Bahkan ketika bentuk kursinya aneh sekalipun



Kita tahu dimana harus duduk

Bahkan ketika itu bukan kursi sekalipun



Kita tahu dari atribut/property yang dimiliki bahwa kita bisa duduk di situ.

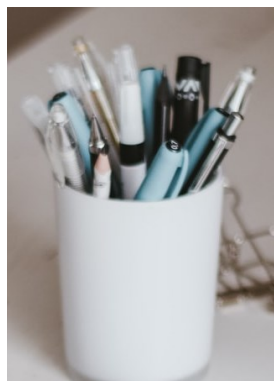
Sama dengan gelas.

Wadah merupakan salah satu atribut/property yang dimiliki gelas.

Dari bentuknya, kita tahu bahwa kalau kita mau minum maka air harus dimasukkan ke wadah itu. Bukan bagian yang lain.



Tapi wadah hanya mengatakan bahwa ia diisi, tapi belum tentu air. Makanya ia bisa diisi alat tulis



dan berat gelas (salah satu atribut/property gelas yang lain) lebih tinggi dari berat kertas, maka ia bisa digunakan sebagai pemberat.

Jadi **AFFORDANCE** antara kita dengan object muncul karena kita menangkap atribut/property tersebut dan memikirkan apa saja yang mungkin kita lakukan dengan atribut/property tersebut.

AFFORDANCE hanya berbicara hal yang mungkin terjadi antara kita dengan object.

Tapi **AFFORDANCE** tidak berbicara hal yang benar dan salah.

Makanya saya bisa memasukkan smackdown dan berdiri di atas kursi sebagai **AFFORDANCE** antara saya dengan kursi. Walau itu bukan cara menggunakan kursi yang benar

Jika tidak ingin disalahgunakan, maka tugas kita sebagai UX Designer adalah menghilangkan **AFFORDANCE** itu dengan cara menghilangkan atau mengubah atribut/property yang bisa disalahgunakan.

Misal, jika kursi kita sekrup ke lantai, maka **AFFORDANCE** untuk diangkat bisa dihilangkan. Sebab atribut/property berat dari kursi berubah dari awalnya bisa diangkat, menjadi tidak bisa diangkat.

Jika memang gelas hanya ingin dipakai untuk menaruh alat tulis, maka kita buat gelasnya bolong. Dengan berubahnya atribut/property dari wadah gelas menjadi bolong, maka gelas tidak mungkin diisi air.



Sama dengan aplikasi/sistem interaktif.

Jika ingin aplikasi/sistem interaktif kita digunakan dengan benar, maka kita perlu atur atribut/property aplikasi/sistem interaktif kita sehingga bisa ditangkap dengan benar oleh user.

Segala atribut/property yang mungkin disalahgunakan, harus dihilangkan/diubah.

Dengan cara itu, maka **AFFORDANCE** yang muncul antara user dengan aplikasi/sistem interaktif akan benar.

Tapi menggunakan **AFFORDANCE** tidak hanya sekedar mengatur atribut/property object.

Dalam bukunya, Norman menjelaskan bahwa **AFFORDANCE** masih bisa dibagi menjadi 4:

- Actual Affordance
- Perceived Affordance

- Hidden Affordance
- False Affordance

Actual Affordance adalah **AFFORDANCE** yang sebenarnya dimiliki object.

Jadi kursi benar-benar bisa untuk diduduki. Gelas benar-benar bisa untuk diisi air.

Sedangkan **Perceived Affordance** adalah **AFFORDANCE** yang orang pikir/percaya dimiliki oleh object.

Jadi Actual Affordance mengambil sudut pandang object, sedangkan Perceived Affordance mengambil sudut pandang seseorang.

Misal kita bicara pintu.

Ada sebuah pintu yang terkunci. Ini adalah Actual Affordance dari pintu itu.

Ketika kalian melihat pintu itu, mungkin ada sebagian yang mengira bahwa pintu itu bisa dibuka. Inilah Perceived Affordance. Kalian *mengira* pintu itu bisa dibuka. Padahal Actual Affordance-nya tidak.

Atau sama seperti cerita Riya.

Ada pintu bertuliskan “Dorong”. Perceived Affordance kalian akan berpikir bahwa pintu itu hanya bisa didorong.

Padahal bisa saja pintu itu memiliki Actual Affordance didorong dan ditarik.

Semoga kalian bisa paham beda Actual Affordance dengan Perceived Affordance.

Lalu ada **Hidden Affordance**, yaitu **AFFORDANCE** yang dimiliki benda tapi orang tidak menyadari.

Misal fungsi lubang kecil pada kaleng soda.

Tidak banyak yang tahu bahwa lubang itu bisa untuk menaruh sedotan



Lubang kecil ini adalah contoh Hidden Affordance.

Termasuk pintu yang terkunci dalam contoh tadi.

Termasuk pintu yang bisa didorong sekaligus ditarik dalam contoh tadi.

Keduanya adalah **AFFORDANCE** yang dimiliki benda tapi orang tidak menyadarinya.

Terakhir adalah **False Affordance**, yaitu konflik antara Perceived Affordance dan Actual Affordance.

Jadi ketika pintu yang terkunci tadi kita buka, disitulah muncul False Affordance akibat konflik antara “pintu bisa dibuka” (Perceived Affordance) dengan “pintu terkunci” (Actual Affordance).

Ketika pintu bertuliskan “Dorong” bisa kita tarik, disitulah muncul False Affordance akibat konflik antara “pintu hanya bisa didorong” (Perceived Affordance) dengan “pintu bisa didorong dan ditarik” (Actual Affordance).

Apa hubungannya keempat jenis **AFFORDANCE** ini dengan UX Design?

Pertama, untuk fitur-fitur yang penting, kita sebagai UX Designer perlu memastikan untuk tidak terjadi Hidden Affordance. Makanya saya tekankan pentingnya **DISCOVERABILITY**.

Kita perlu membuat user sadar bahwa ada **AFFORDANCE** penting yang bisa dilakukan dengan aplikasi/sistem interaktif tersebut.

Kedua, kita perlu meminimalisir jumlah False Affordance. False Affordance terjadi karena apa yang dipikirkan user tidak sama dengan kenyataan. Ini adalah konsep **Gulf of Execution**.

Makanya kita perlu mengatur atribut/property dari aplikasi agar bisa meminimalisir penyalahgunaan akibat False Affordance tadi.

Bagaimana caranya?

Pertama, pastikan aplikasi memang memiliki Actual Affordance tersebut.

Aneh saja kita bilang aplikasi/sistem interaktif bisa kirim email tapi ternyata dia tidak punya fitur kirim email. Artinya sejak awal, aplikasi/sistem interaktif memang tidak punya Actual Affordance kirim email.

Kedua, kita bisa menggunakan **SIGNIFIER**, **CONSTRAINT**, **FEEDBACK**, dan **MAPPING** sebagai alat bantu untuk meminimalisir Hidden Affordance dan False Affordance.

Signifier

Jadi saya sudah jelaskan pentingnya membantu user membangun **Mental Model** dengan **System Image** yang sederhana, meningkatkan **DISCOVERABILITY**, dan meminimalisir Hidden Affordance dan False Affordance.

Tapi bagaimana cara spesifik untuk mencapai semua itu?

Nah **SIGNIFIER**, **CONSTRAINT**, **FEEDBACK**, dan **MAPPING** adalah principle yang akan membantu tercapainya semua itu.

Kita mulai dari **SIGNIFIER**.

SIGNIFIER berasal dari kata Sign, yang artinya adalah tanda atau petunjuk.

Sesuai dengan namanya, **SIGNIFIER** berfungsi untuk memberi petunjuk kepada user agar menggunakan object dengan benar.

Petunjuk apa?

Petunjuk yang diberikan **SIGNIFIER** adalah petunjuk:

- Apa
- Di mana
- Bagaimana

SIGNIFIER Apa misalnya...



Tulisan “EXIT” menjadi **SIGNIFIER** ini pintu Apa



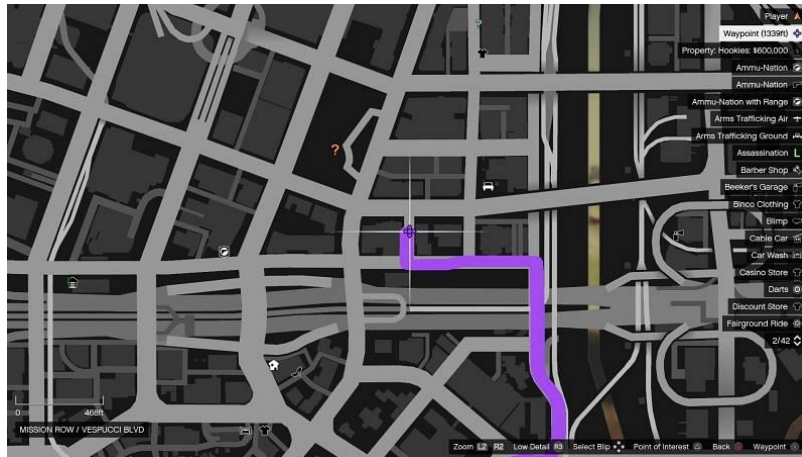
Tulisan “Penelitian” menjadi **SIGNIFIER** ini halaman Apa

SIGNIFIER Di mana misalnya...



Tulisan “Pria”, “Wanita” dan “Toilet” merupakan **SIGNIFIER** Apa

Tapi arah panah merupakan **SIGNIFIER** Di mana



Petunjuk arah di game GTA merupakan **SIGNIFIER** Di mana

SIGNIFIER Bagaimana misalnya...



Tulisan “Pull” dan “Push” menjadi **SIGNIFIER** mengenai Bagaimana cara menggunakan pintu

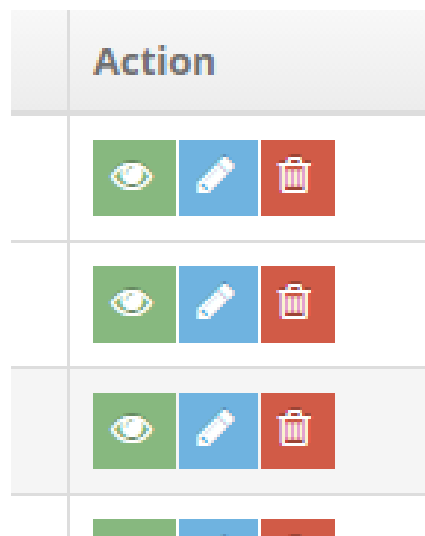


Tulisan “Insert coin to continue” menjadi **SIGNIFIER** mengenai Bagaimana cara memulai game.

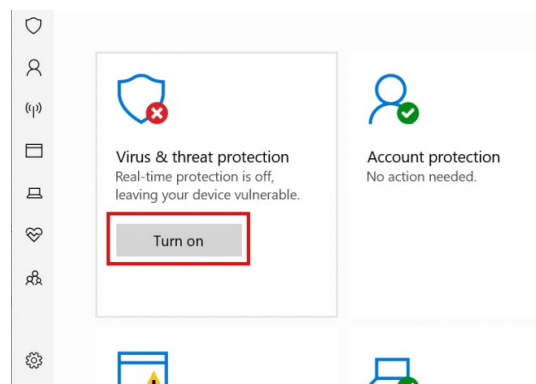
Dengan memberi **SIGNIFIER**, kita berharap bisa mempermudah **Mental Model**, meningkatkan **DISCOVERABILITY**, dan meminimalisir Hidden Affordance dan False Affordance.

Tapi yang perlu kalian sadari, **SIGNIFIER** tidak melulu harus text.

Kita bisa menggunakan icon sebagai **SIGNIFIER**



Atau warna sebagai **SIGNIFIER**



Atau suara, bahkan animasi/gerakan sebagai **SIGNIFIER**

Jadi jangan berpikiran sempit bahwa **SIGNIFIER** pasti harus berbentuk text.

Constraint

SIGNIFIER utamanya memberi petunjuk cara menggunakan yang benar.

SIGNIFIER bisa saja memberi petunjuk untuk mencegah terjadinya kesalahan

Tapi **SIGNIFIER** hanya bisa memberi petunjuk, ia tidak benar-benar bisa mencegah.

Di sinilah muncul **CONSTRAINT** untuk membantu.

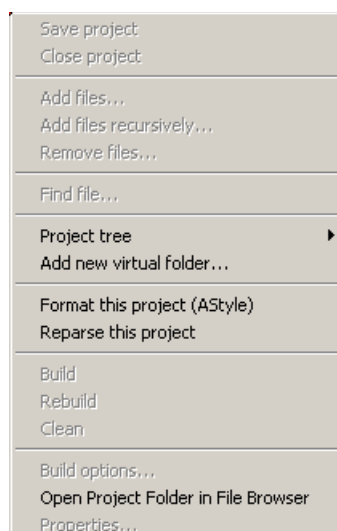
Sesuai dengan namanya, **CONSTRAINT** berfungsi untuk menghalangi atau membatasi interaksi user.

Dengan **CONSTRAINT**, kita bisa meminimalisir user melakukan tindakan yang salah.

Misal port pada laptop atau PC



Adanya **CONSTRAINT** pada bentuk port memastikan bahwa tidak mungkin salah memasukkan Plug/Connector



Adanya **CONSTRAINT** fitur yang tidak bisa diklik memastikan bahwa fitur tersebut tidak bisa digunakan kecuali pada saat yang tepat

Adanya **CONSTRAINT** pada pemilihan jenis kelamin memastikan bahwa user hanya memilih antara laki dan perempuan. Bayangkan kalau dibuat textbox (kembali ke bab 2).

Selain itu, adanya **CONSTRAINT** pada pemilihan tanggal juga memastikan user tidak asal memilih tanggal lahir. Bayangkan jika dibikin textbox, akan ada banyak variasi bentuk tanggal yang bisa diisi.

Adanya **CONSTRAINT** step 1-6 memaksa user untuk tidak bisa langsung loncat ke langkah terakhir saat membeli barang. Agar bisa menuju langkah terakhir, user harus mengikuti setiap langkah dari 1-6 step by step.

Kombinasi antara **SIGNIFIER** dan **CONSTRAINT** akan membantu user meminimalisir **GULF OF EXECUTION**.

Feedback

Dalam pengalaman saya, melakukan kesalahan saat menggunakan aplikasi/sistem interaktif bukan hal yang seram.

Yang seram adalah aplikasi yang tidak memberikan respon sehabis kita melakukan aksi.

Bayangkan saja kalian naik lift dan kalian menekan tombol sebuah lantai.

Setelah menekan, pintu tidak menutup, tombol tidak menyala, dan lift tidak bergerak.

Kalian mungkin akan bingung dengan apa yang terjadi

Apa liftnya rusak?

Apa saya kurang keras menekannya?

Atau ...???

Atau bayangkan kalian lagi ingin memindahkan file-file film animasi kalian dari PC ke hard disk eksternal.

Setelah melakukan proses cut/copy file, kalian tidak melihat ada perubahan di layar PC kalian. Tidak ada progress bar, tidak ada folder yang melemparkan isinya ke folder lain, tidak ada pergerakan sama sekali.

Ini file sudah dipindah? Sedang dipindah?

Ini apa komputer saya freeze?

Atau ...???

Seperti yang saya sampaikan sebelumnya, kita bisa mencoba semua tombol yang ada di depan kita.

Bisa juga sebuah tombol sudah diberikan **DISCOVERABILITY** sekaligus **SIGNIFIER** yang bagus.

Tapi jika aplikasi/sistem interaktif tidak memberikan reaksi apapun ketika tombol ditekan, maka **UNDERSTANDING** user tidak akan berkembang.

Inilah pentingnya memiliki **FEEDBACK** yang baik dalam aplikasi/sistem interaktif.

FEEDBACK yang baik akan membantu user membentuk **Mental Model**-nya, yang ujung-ujungnya adalah meningkatkan **UNDERSTANDING**.

Bagaimana **FEEDBACK** yang baik?

Ciri-ciri **FEEDBACK** yang baik:

- Langsung
- Menunjukkan kondisi aplikasi/sistem interaktif yang sebenarnya
- Menunjukkan jika tindakan user benar/salah
- Bisa ditangkap oleh user
- Bisa dipahami oleh user

Langsung dalam artian bahwa **FEEDBACK** harusnya merupakan reaksi langsung dari tindakan user. Jadi tidak boleh ada delay.

Delay sedikit saja terkadang sudah bisa membuat bingung.

Jika memang aksi dari user membutuhkan waktu yang tidak sebentar, misal seperti proses memindah file besar dari satu folder ke folder lain, aplikasi/sistem interaktif tetap harus memberikan **FEEDBACK** bahwa proses sedang berjalan.



Menunjukkan kondisi aplikasi/sistem interaktif yang sebenarnya dalam artian bahwa **FEEDBACK** tidak boleh salah atau berbohong.

Jika lift sedang berada di lantai 2, harusnya lampu yang menyala ya lampu lantai 2

Jika file sudah pindah 50%, harusnya memang file yang pindah ya 50%

Jika ada notifikasi bahwa file sudah di-delete, harusnya memang file sudah tidak ada lagi.

Jika **FEEDBACK** bohong atau salah, maka user akan merasa dibohongi, dan ini bisa menurunkan **UNDERSTANDING** user terhadap aplikasi/sistem interaktif.

Menunjukkan jika tindakan user benar/salah dalam artian bahwa harus ada perbedaan antara **FEEDBACK** ketika suatu tindakan benar dengan **FEEDBACK** ketika suatu tindakan salah.

Makanya **FEEDBACK** dapat dibedakan menjadi tiga jenis:

- Positive Feedback → **FEEDBACK** sebagai reaksi terhadap tindakan user yang benar
- Negative Feedback → **FEEDBACK** sebagai reaksi terhadap tindakan user yang salah
- Progress Feedback → **FEEDBACK** sebagai informasi bahwa proses belum selesai

Contoh Positive Feedback misalnya halaman masuk ke home setelah login dengan username dan password yang benar

Contoh Negative Feedback misalnya halaman tidak berpindah ketika login dengan username atau password yang salah. Selain itu, biasanya ada pemberitahuan bahwa login salah.

Contoh Progress Feedback misalnya progress bar atau loading animation.

Dalam UX Design, biasanya mahasiswa saya terlalu fokus pada Positive Feedback sehingga lupa mendesain Negative Feedback dan Progress Feedback.

Bisa ditangkap oleh user dalam artian bahwa **FEEDBACK** juga harus bisa disadari atau diterima oleh user.

Makanya **DISCOVERABILITY** tidak hanya membahas fitur yang mudah ditemukan, tapi juga membahas **FEEDBACK** yang mudah ditemukan.

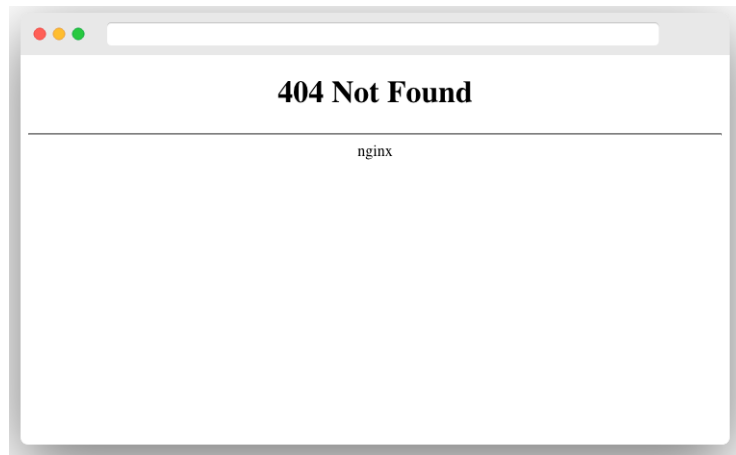
Jika **FEEDBACK**-nya ada tapi user tidak sadar, ya percuma saja. **UNDERSTANDING** user tidak akan meningkat.

Makanya tadi saya bilang aplikasinya seram. Karena cuma diam aja.

Terakhir, **Bisa dipahami oleh user** dalam artian bahwa user memahami apa arti dari **FEEDBACK** tersebut.

Jika **FEEDBACK** sudah diberikan, user sudah menangkap, tapi user tidak paham dengan **FEEDBACK**-nya, ya tetap percuma.

Misal...

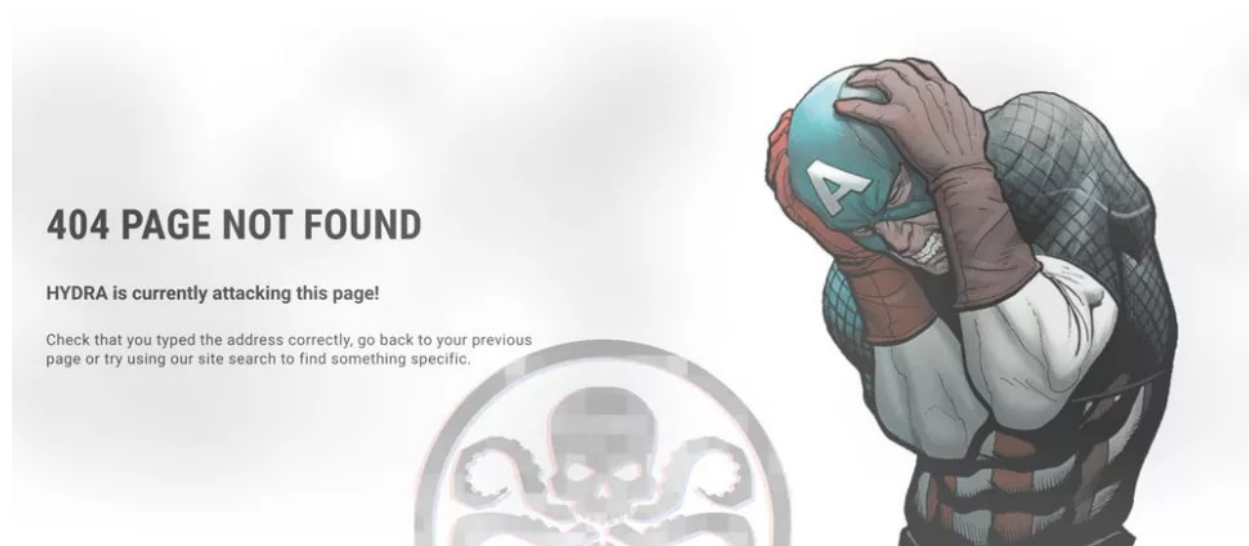


Jika kalian bukan anak IT, kira-kira bisa paham ga arti dari **FEEDBACK** tersebut?

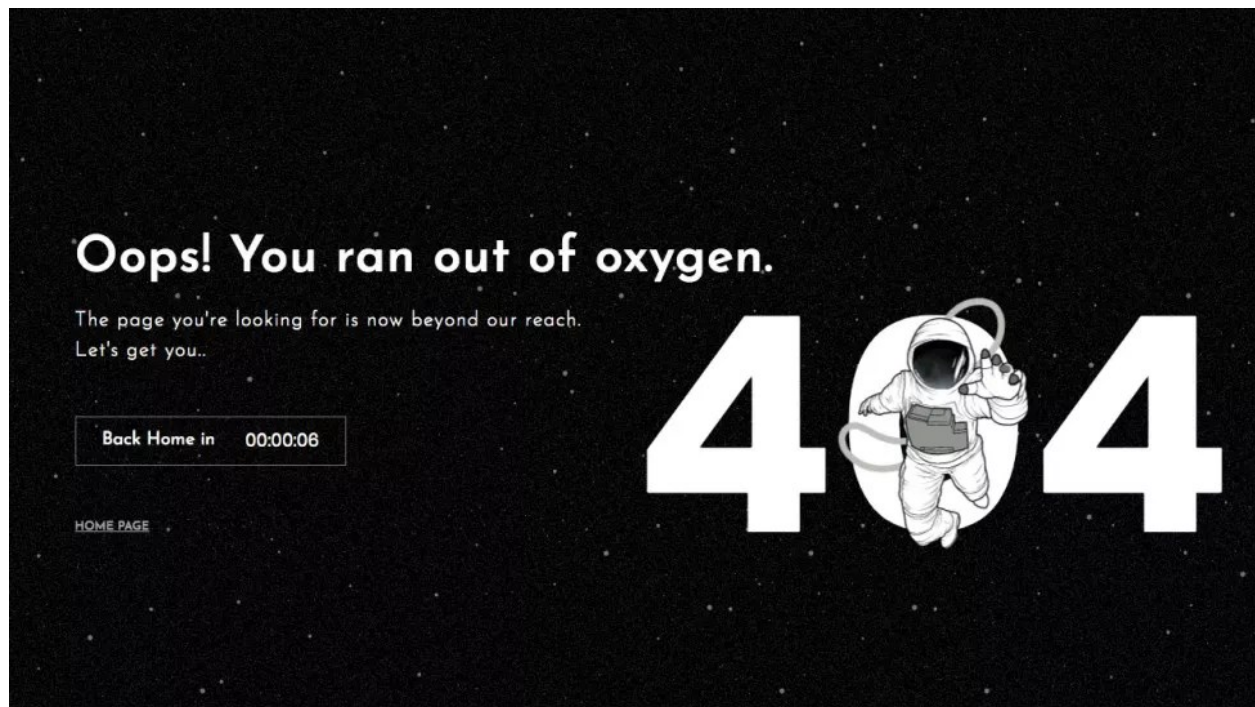
Jadi **FEEDBACK** sudah diberikan, kita pun sudah baca errornya, tapi kita tidak paham dengan errornya.

Makanya halaman itu adalah contoh **FEEDBACK** yang buruk.

Bandingkan dengan



Atau dengan



Selain lucu, **FEEDBACK** ini juga lebih mudah dipahami.

Dan itulah cara menggunakan **FEEDBACK** yang baik.

Terkhusus Negative Feedback, sebuah Negative Feedback yang baik memiliki 3 syarat:

- Saya tahu ada yang salah
- Saya tahu salahnya karena apa
- Saya tahu bagaimana cara membenarkan kesalahan saya

Jika Negative Feedback membuat user ragu apakah terjadi kesalahan atau tidak, maka itu adalah Negative Feedback yang buruk. Makanya biasanya digunakan warna merah, tanda silang, atau tulisan error yang besar untuk jelas-jelas menunjukkan bahwa ada kesalahan.

Tapi jika user hanya tahu bahwa ada yang salah, itu tetap tidak akan maksimal dalam membentuk **Mental Model**-nya.

Agar lebih sempurna, user perlu tahu siapa yang salah dan salahnya karena apa.

Ketika terjadi kesalahan, insting user akan selalu mengatakan bahwa ia yang melakukan kesalahan. Padahal belum tentu.

Kesalahan bisa terjadi karena faktor di luar user. Misal jaringan putus, internet lambat, listrik mati, bahkan maintenance.

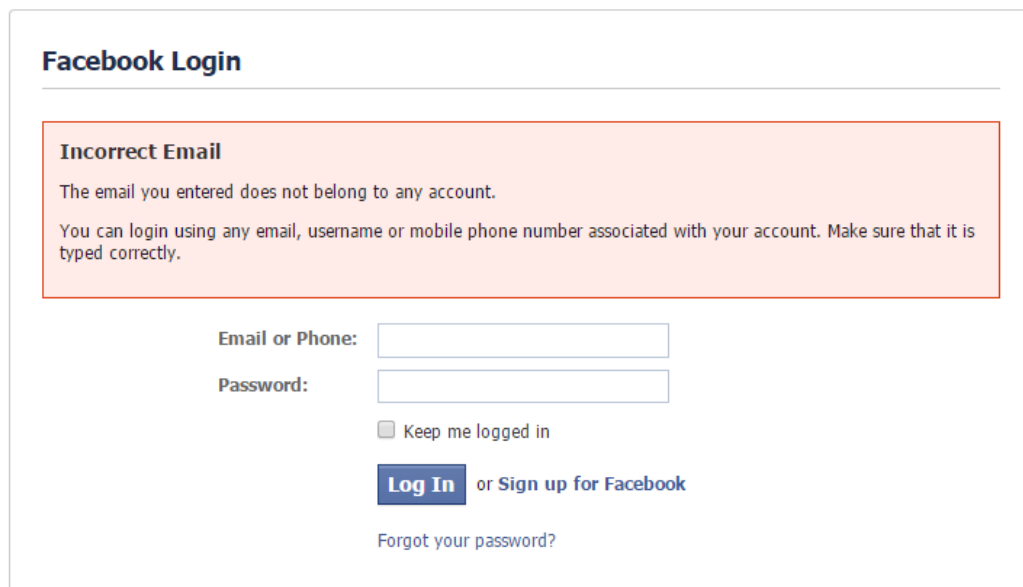
Jadi Negative Feedback yang baik perlu menunjukkan siapa yang melakukan kesalahan.

Nah, seandainya user yang melakukan kesalahan. Maka ia perlu tahu apa tindakannya yang menyebabkan terjadinya kesalahan. Dengan begitu, ke depannya user bisa lebih berhati-hati.

Tapi dalam aplikasi/sistem interaktif, terkadang user tidak mengetahui bagaimana cara untuk menghindarinya atau memperbaikinya. Akibatnya, bisa saja dia stuck atau mengulang-ulang kesalahan karena tidak tahu cara yang benar seperti apa.

Oleh karena itu, Negative Feedback yang baik akan memberitahu user bagaimana caranya membenarkan/memperbaiki kesalahan.

Misal...



The screenshot shows a Facebook Login interface. At the top, it says "Facebook Login". Below this, there is a red-bordered box with the title "Incorrect Email". Inside the box, the text reads: "The email you entered does not belong to any account. You can login using any email, username or mobile phone number associated with your account. Make sure that it is typed correctly." Below the error message, there are two input fields: "Email or Phone:" and "Password:". Below the "Email or Phone:" field, there is a checkbox labeled "Keep me logged in". Below the "Password:" field, there is a blue button labeled "Log In" followed by the text "or Sign up for Facebook". At the bottom, there is a link that says "Forgot your password?".

Dalam contoh tersebut terlihat jelas bahwa ada kesalahan yang terjadi. Baik itu dari warna merah atau dari tulisan "Incorret Email"

Selain itu, contoh tersebut juga menunjukkan siapa yang salah dan apa kesalahannya. Bisa dibaca dari kalimat "the email you entered does not belong to any account"

Terakhir, di contoh tersebut juga disebutkan bagaimana cara untuk memperbaikinya. Salah satunya dengan kalimat "Make sure that it is typed correctly".

Jadi bisa dikatakan, itu adalah contoh Negative Feedback yang baik.

Sekali lagi, jangan meremehkan **FEEDBACK** dalam aplikasi.

Salah satu kunci untuk menutup **Gulf of Execution** ada pada **Gulf of Evaluation**.

Dengan **FEEDBACK** yang baik, otomatis **Gulf of Evaluation** akan menyempit.

Dengan demikian, akan ada perkembangan pada **Mental Model** user.

Mapping

Principle yang terakhir adalah **MAPPING**.

MAPPING merupakan relasi antara remote control/controller dengan object yang dikontrol.

Berbeda dengan **AFFORDANCE** yang merupakan relasi antara seseorang dengan object.

Dalam **AFFORDANCE**, interaksi yang kita lakukan dengan object biasanya terjadi secara langsung.

Ketika saya duduk di kursi, maka saya langsung duduk di kursi

Ketika saya mengangkat kursi, maka saya langsung mengangkat kursi

Tapi ketika saya menyuruh orang lain mengangkat kursi, maka saya tidak mengangkat kursi secara langsung. Orang lain itulah yang mengangkat kursi

Konsep **MAPPING** kurang lebih seperti itu.

Ketika kita ingin menggunakan produk, terkadang kita diberikan remote control atau controller sebagai perantara untuk menggunakannya.

Misal perantara AC adalah remote AC

Perantara lampu ada tombol lampu

Perantara komputer adalah mouse dan keyboard

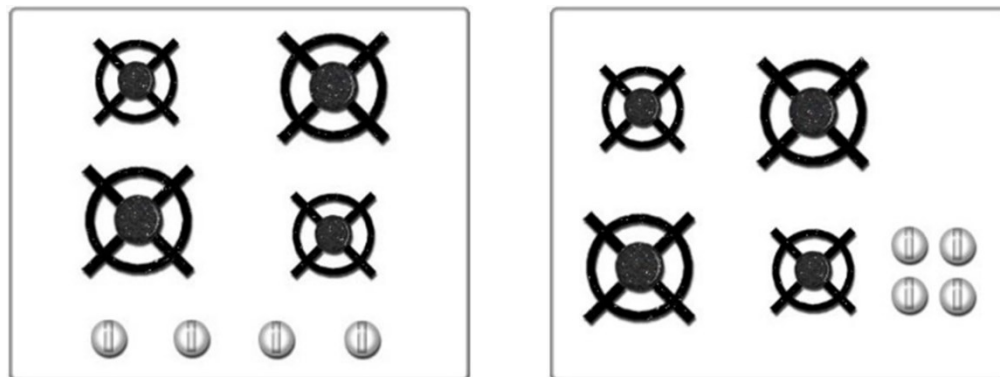
Perantara aplikasi adalah tombol, drop down, dan UI element lainnya

Artinya, untuk bisa menggunakan AC, kita hanya bisa melakukannya melalui remote AC. Sama seperti ketika ingin menggunakan lampu, kita hanya bisa melakukannya melalui tombol lampu.

Agar kita bisa memahami **Mental Model** dari object yang dikontrol, perlu ada mapping yang baik dengan controller-nya.

Contoh yang sering digunakan Norman adalah kompor

Menurut kalian kompor mana yang lebih mudah dipahami cara kerjanya?



Di kompor yang kanan, **Mental Model** akan lebih mudah terbentuk. Sebab ada **MAPPING** yang jelas antara letak kompor dengan letak tombol controllernya.

Di kompor yang kiri, **MAPPING**-nya tidak terlalu jelas karena posisi kompor dan posisi tombol controller-nya kurang berelasi.

Mungkin kalian akan berpikir “Kan bisa dikasih tulisan pak?”

Benar!

Bisa saja dikasih tulisan tombol mana untuk kompor yang mana.

Tapi itu tugas **SIGNIFIER**.

Dalam **MAPPING**, relasi harusnya bisa ditunjukkan tanpa banyak menggunakan **SIGNIFIER**.

Makanya kompor kanan adalah contoh **MAPPING** yang baik, karena kita bisa mendapatkan **UNDERSTANDING** tanpa perlu menambah **SIGNIFIER**.

Sama seperti tombol lampu dan lampu.



Jika **MAPPING**-nya baik, harusnya tombol kanan untuk lampu kanan dan tombol kiri untuk lampu kiri.

Atau keypad untuk berjalan/bergerak saat main game PC



Mengapa tombol WASD yang dipilih?

Kenapa bukan QWER?

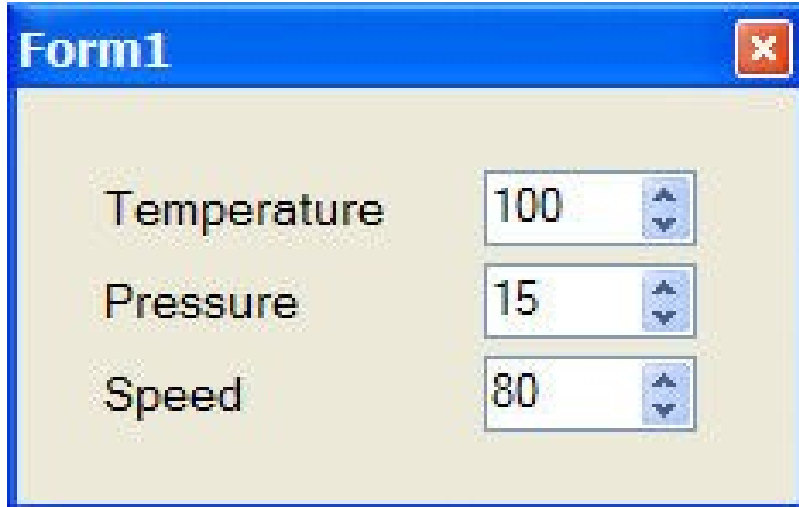
Kenapa bukan ASDF?

Karena WASD memberikan **MAPPING** yang lebih baik. Jelas mana tombol atas-bawah-kiri-dan kanan.

Kalau menggunakan ASDF, mungkin kiri dan kanan akan gampang ketahuan.

Tapi bagaimana dengan atas dan bawah?

Sama juga dengan ...



Kalau mau menaikkan, **MAPPING**-nya adalah ke atas

Kalau mau menurunkan, **MAPPING**-nya adalah ke bawah

Itupun antara controller panah atas bawah dan object yang diubah juga diletakkan berdekatan. Jadi jelas controller mana untuk object yang mana.

Itulah contoh-contoh **MAPPING** yang baik.

Dengan **MAPPING** yang baik, kita bisa membantu **UNDERSTANDING** hanya dengan menggunakan bentuk dan letak. Tidak perlu menambah **SIGNIFIER** jika memang tidak diperlukan.

Rangkuman

1. User sulit berinteraksi dengan aplikasi karena munculnya **Gulf of Execution** dan **Gulf of Evaluation**
2. Menurut Norman, untuk mengatasi **Gulf of Execution** dan **Gulf of Evaluation**, kuncinya ada pada **GOOD DESIGN**
3. Untuk menghasilkan **GOOD DESIGN**, kuncinya adalah **DISCOVERABILITY + UNDERSTANDING**
4. Untuk meningkatkan **DISCOVERABILITY** dan **UNDERSTANDING**, Norman mengenalkan **SEVEN FUNDAMENTAL DESIGN PRINCIPLE**
5. **SEVEN FUNDAMENTAL DESIGN PRINCIPLE** tersusun atas:
 - a. **CONCEPTUAL MODEL**
 - b. **DISCOVERABILITY**
 - c. **AFFORDANCE**
 - d. **SIGNIFIER**
 - e. **CONSTRAINT**
 - f. **FEEDBACK**
 - g. **MAPPING**

6. **CONCEPTUAL MODEL** berarti membantu pembentukan **Mental Model** menggunakan **System Image** yang sederhana
7. **DISCOVERABILITY** berarti user harus mudah menemukan apa saja yang bisa ia lakukan, dimana ia bisa melakukannya, dan bagaimana melakukannya
8. **AFFORDANCE** berarti perlu bijak dalam mengatur atribut/property object agar meminimalisir Hidden Affordance dan False Affordance
9. **SIGNIFIER** berfungsi untuk memberi petunjuk kepada user agar menggunakan object dengan benar
10. **CONSTRAINT** berfungsi untuk menghalangi atau membatasi interaksi user
11. **FEEDBACK** berarti memberikan reaksi yang langsung, nyata, mudah diterima, dan mudah dipahami setelah ada tindakan dari user
12. **MAPPING** berarti memberikan relasi yang natural antara remote control/controller dengan object yang dikontrol sehingga user bisa memahami relasi tersebut tanpa bantuan **SIGNIFIER**