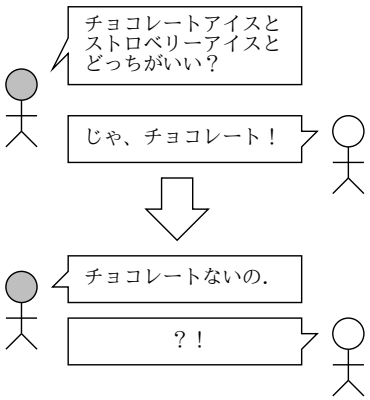
Desigining from Both Sides of the Screen: 1章

久野　靖*

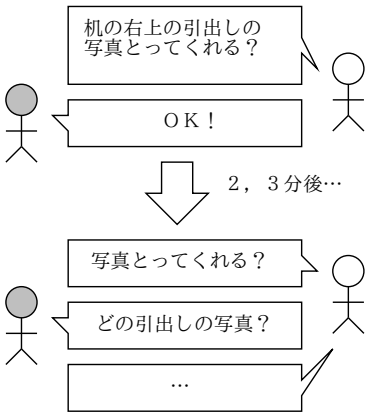
2004.9.6

0　Introduction

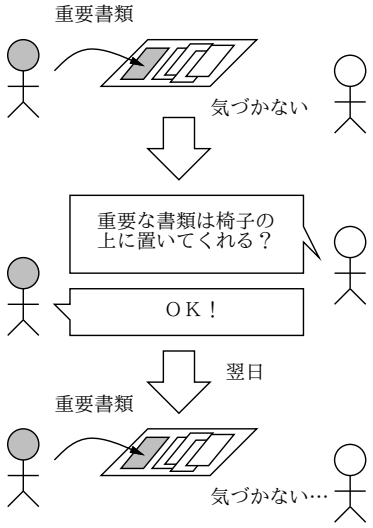
□ たとえば…



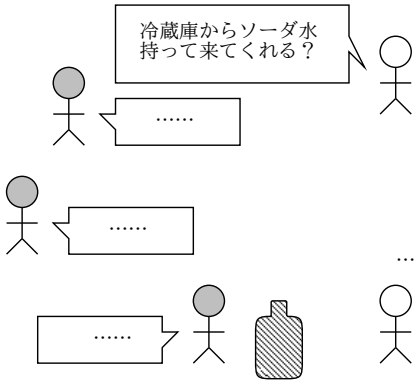
□ たとえば…



□ たとえば…



□ たとえば…



□ これらはすべて「非協力的な態度」といえる。

- 友人がいつもこういう風だったら、そいつとは友達でいたくないと思う。
- しかしコンピュータは…いつもこういう風でアタリマエ、皆もそれで普通だと思っている。なんで???

□ コンピュータがいつも「やってくれる」こと…

- 選択肢が提示されて、希望を選ぶと後で「それは駄目」と言われる。

*筑波大学大学院経営システム科学専攻

- ファイルを開こうとするといつも「標準フォルダ」から選ばそうとする (つい 1 分前にアクセスしたフォルダが正解なのに…)。
- 窓が出て来て、邪魔だからどかすんだけど、またすぐ同じ邪魔な場所に窓が出て来る。
- 指示を出しても何もフィードバックがなく、もっかい指示したものかどうか悩んでしまう。
- まだまだ、たくさん…

□ 技術開発に携わるなら、製品に「ユーザと協力的になる」ことを教えるべきである。

- 技術は人々に対しもっと nice でなければいけない。
- 人どうしが健康で協力的な関係を築いている時に従っている「規則」を取り上げ、技術製品も同じ規則に従うようにする。

□ 技術開発の場合、「やっていいこと」「いけないこと」があれば十分というわけではない。

- →製品を作るプロセスにそって、どうやればいいか、それはなぜかを学ぶ。
- 我々(著者) は 2 人→デザイナと技術者→目標は、頑健で、信頼性があり、応答が速く、メンテしやすい「ことに加え」、使えて、役に立つ製品を作ること。
- 実際に我々が作った製品を見てもらってもいい。http://www.HubbubMe.com

0.1 Who We Are

□ Ellen → interaction designer, Alan → software engineer.

- 経験 15 年。ソフト開発、Web アプリ開発、Palm アプリ等。

□ Ellen →ユーザ中心開発、使いやすさに関心。コードは書かない (大学で Pascal、あと Web スクリプト)、何が難しいかといったことは分かる。

□ Alan → Unix hacker、アーキテクチャや内部機能に関心、ユーザインタフェースは「皮」。頑丈で、性能がよく、保守しやすいソフトに注力。よい UI が大変なことにいつも驚かされる。

□ この 2 人の視点から協力することでよいものができる→本書を書いた動機。

0.2 Who You Are

□ 技術開発プロセスに参加している複数タイプの人に有用な本。

□ ソフト技術者で、開発するものの使いやすさを改善したい (けど、具体的にどうしたらいいか分からない) 人。とくにアプリケーション開発、UI ツールキット開発の人。自力でデザインする場合と、UI デザイナと協力して開発するのでデザイナの仕事を知りたい場合とがある。

□ UI デザインプロセス/テストプロセスに関与する人。interaction designer(UI デザイナ) で、デザインをもっといいものにし、技術者ともっと効果的に仕事したい人。グラフィックデザイナ (visual designer) で、ユーザインタフェースの構造や動作について学び、ビジュアルデザインとの関わりを知りたい人。usability testing specialist(usability engineer) で、デザインプロセスについて学んだり、利用調査の方法を知りたい人。

□ ソフトウェア技術マネージャ、デザインマネージャで、ユーザに受け入れられる製品を作りたい人、ユーザの不満のためリリースごとに大幅手直しするのをやめたい人。

□ マーケティングマネージャ、製品開発マネージャで、自分達の製品がなぜそんなに使いにくいのか、それを改善するにはどうしたらいいか知りたい人。

□ テクニカルライターで、デザインプロセスや技術者との協同作業について知りたい人。

0.3 What This Book Covers

□ ソフトウェア開発に関するよい本は沢山ある。Steve McConnell の Rapid Development、Steve Maguire の Debugging the Development Process をお勧め。

- しかしソフトウェア工学の本は UI デザインやその開発プロセスとの統合についてはちょっとしか触れていない。

□ UI デザインに関するよい本も沢山ある。Jeff Johnson の GUI Bloopers、 Steve Krug の Dont Make Me Think、Jakob Nielesen の Designing Web Usabiliy、Eric Bergman の Information Appliances and Beyond がお勧め。しかしデザイン指針の技術面への関与についてはほとんど触れていない。

□ ソフト開発の本も、UI デザインの本も、自分の領分についてどうすべきかは述べているが、いちばん難しい両者のトレードオフについては触れていない。

- また、実際のインタフェースがデザインされ、構築され、調査され、繰り返しテストされ、使いやすいものになる過程を具体的に示した本もない。
- 本書はこれらの面を提供する。

□ 本書では読者が「技術は使いやすくなければいけない」という前提を受け入れるものとしている。この点については、Alan Cooper の The Inmates Are Running the Asylum、Don Norman の The Design of Everyday Things を勧める。

- もし、一般には技術は使いやすくなければならないと思うが、それに掛かる労力に見合うかどうか自信がないなら、本書が役立つかも。
- 本書では労力と効果を直接比較はしていないが、労力を増やさずに使いやすいソフトウェアを作る方法を提示する (魔法ではない)。
- なぜこれがよいか、またどうやって機能を選択するか説明する。

0.4 Part I: The Goal

□ どのような目標を達成しようとするかについて。

- 協調性の原理と、そのよい例、よくない例をさまざまな方面の例をあげて見て行く。
- よい、協調的なインタフェースについてそれと分かるようになる。

0.5 Part II: The Process

□ 上記のようなものを作るプロセスについて学ぶ。

- Hubbub という実際のアプリケーションを例として、プロセスを順に詳しく見て行きながら学ぶ。
- 抽象的な例でなく本もののソフトを最初から最後まで対象とすることで、現実に起こる複雑な判断について知る。
- これらの複雑な判断…デザインルール間のトレードオフ、デザインと技術の間のトレードオフ…が、品質の決め手になる。
- これにより、抽象論の本を実地に適用するにはどうするかも分かる。
- 逆に取り上げたアプリのあれこれに立ち入りすぎてしまう危険もある…ので、まず基本原理について (第 1 部で) 学んでから。

1 On Being a Butler

□ 何かある好きなことに熱中している時 (テニス、写真、ギター、バイク、…) →「流れるように」操作する

- 何か道具に問題が起きると (靴紐がほどける、フォーカス速度が遅くなる、チューニングがずれる、アクセルが引っかかる、…) →ちょっとした引っかかりでも、「流れ」は壊されてしまう。

□ 道具の「引っかかり」→道具の問題を直そうとするが、ある程度やっても駄目ならその道具を取り換えてしまう。

- あたかも、道具に対する「忍耐ポイント」が (その道具の持つ利点に応じて) 与えられてるかのよう。
- 「流れ」が邪魔されるたびにポイントが減算されていく。利点より減点が上回るとその道具を使わなくなる。
- 多くの人にとって、コンピュータ技術はこの「分岐点すれすれ」。確かに今までできなかったことが色々できるが、使ってて楽しくないし、イライラすることが多い。

□ なぜそんなことに? 製作者たちは「いいもの」「使って楽しいもの」を作りたいと思っているはずなのに…

- 筆者が思うに、その原因は (「使いやすさ競争」ではなく)「機能の数競争」に陥っているから。
- 開発者は「機能リスト」をつくり、営業マンはそれをもとに営業し、レビュー者 (雑誌等) は機能の有無をマトリクスで表示して比較。そのような世界では「機能が多いほどいい」になってしまう。
- 「使いやすさ」を機能のうちに挙げることもあるが、明確な基準がないので単に GUI がついてる、Web インタフェースがある、程度で、それだけで勝負になるわけではない。

□ 一方で、マスコミも雑誌も「技術製品は複雑すぎて使いこなせない」と言い続けている。1991 年くらいからずっと。

- どれも同じこと…「機能が多すぎて、使いやすさに注意が払われていない」を書いている。
- つまり、機能を減らすのは一見リスキーに見えるが、そうすることで使いやすくできればユーザはついてくるという証拠がある。

□ Alan Cooper は Apple がそうだと言っている。Apple 製品は高くてアプリが少いが、ファンは浮気しない。

□ Palm Pilot 以前からハンドヘルド PC の試みはあった (Go、Newton、…) が、どれも盛大に発表したものの失敗。今日、Palm は成功をおさめ称賛されている。

- そうするためには、「流れにのって、スムーズに」使えるようにすることが重要。「流れる」ことは、機能自体と同じくらい重要。

□ 多くのプロジェクトは機能のリストを作って完成時に「ある/なし」をチェックする。そこに「使いやすさ」という機能を含めるようにして、チェック時に「他の機能全般にわたってひっかかりがないこと」をチェックしよう。それが OK でない限り、元の機能も OK でない。

□ 開発開始時に、技術者、デザイナ、営業、マネジメントの全員がこの「使いやすさ」に時間とエネルギーを費して取り組むことを確認するべき。

- それはつまり、「多くの人が使いたいと思うもの」を作るか、「1 度だけ試してみて不平を言うもの」を作るかの分かれ目。

1.1 It's the Relationship, Stupid

□ 「役に立ち、使えるものを作る」ことは受け入れたとして、ではどうするか?

□ 技術製品を作るとき、「機能の詰まったツール」を作るわけではないと肝に命じる。

□ そうではなく、「ユーザと関わりを持つ何か」を作る!

- 人の操作に応答する。人が何をするかによって動作が変わる。人のために何かをする。どういう風にやるか尋ねさえするかも。よりよくやるために、情報を求める。時としてミスもして、何がうまく行かなかったためか説明する。
- 人の方は、その製品が何をおこなえ、どうやって仕事を頼むかを学ぶ。その製品の応答に応じて行動を変える。その製品のために喜んだり、イライラしたりする。

□ → relationship 「関わり」

□ 深い関わりというわけではないし、対等というわけでもない。

- 人のために何かを提供する。その反対ではない。
- しかし関わりであるから、人は多くの「適切なふるまい」に関する期待を持つようになる。
- その「期待」を理解すればするほど、その関わりを喜ばしいものにできる。

□ よいモデル: 執事と主人。もちろん執事を直接知っている人はあまりいないが、映画などを見ればよい執事はこういう感じ、というのは想像できる。

- 執事はいつでもそこにいて、何か頼まれたらすぐやる用意ができている。質問は少し、不平はなし。
- 何か問題があれば、主人を煩わさずに回避する方法を見つける。主人を煩わさないことは重要なので、何ができますよと自分から言い出すことはめったにない。その代わり、これまで主人がやってきたことをよく見ていて、主人が将来どういうことをして欲しそうか分かろうとする。
- しかし、先回りしてやりすぎない。主人がやって欲しいと思わない方法で何かをやってしまうくらいなら、自分からやり出さない方がよい。
- 能力と同じくらい自分の態度、ふるまいも重要だと知っているので、丁寧でうやうやしくあろうとする (主人が自分にできないことを指示した場合でも)。

□ 主人は執事が何ができ何が得意かを観察する。どうやって指示を出すのがいいか尋ねたりはしない。執事の反応を見ながら、徐々に、どういう風に指示するのがよい結果をもたらすか学んでいく。

- 時がたってよい関係ができあがれば、互いのやりとりを意識せずに一緒に働けるようになる。協調することができる。

□ …というわけで、技術製品を作るときの目標は、その製品に執事のようにふるまう方法を教えること。そのためには、どうやって協調するかを教えること。

1.2 How to Collaborate

□ すべての社会活動→「適切なふるまい」の慣習を持つ。会話、ダンス、バンドでの演奏、ドライブ、家の建設、一緒に食事。いずれも慣習に従う。

- 慣習はふつう、陽に語られることはない。時として、たまたまそういう風だったりする。しかしそれにより、他人のふるまいが予想でき、うまく一緒に行動できる。
- 「丁寧なふるまいの規則」と考えてもよい。ドライブなら、レーン変更したいときウィンカーを出す。劇の終わりには拍手する。話したいときは挙手する。

- 協調とは、このような慣習への適合。それによって、活動の興味深い部分に注意を注ぐことができる。

□ 社会科学者は何年にも渡って「協調行動」について調べてきた。特に会話に関係するもの。→その成果を技術製品とユーザが協調するための規則のために活用したい。

- 丁寧さ→プラスの丁寧さ、マイナスの丁寧さ。

□ マイナスの丁寧さ→害をなさない、いろいろ質問しない、主張しない、攻撃しない。

- 技術製品はどうか? 少しの利得のために、多くの努力を要求する。多くの選択肢を示すので、ユーザはしたいことが見つけられない。ユーザにあまりに多くのことを憶えておくよう要求する。
- だから最低限、製品にはマイナスの丁寧さを持たせる→粗野に見えないようにする。これ自体、多くの製品は達成し損なっているので、これに成功すればあなたの製品は特徴あるものとなる。

□ ユーザに喜んでもらうには、プラスの丁寧さ→積極的な協調

- どうすればいいか全部書き記すのは難しいが…

□ 1967, Paul Grice(言語学者) → cooperative principle for conversation → Quantity, Quality, Relevance, Manner.

□ Quantity: その状況に適切な情報を、多すぎず、少なすぎず。

- 「小切手は OK ですか?」→×「いいえ」○「いいえ、クレジットカード、デビッドカード、現金を頂いてます」
- なぜその方がいいか→あなたの「支払う (たぶん現金以外で)」という目的にアプローチ、適切な代替案を提示。
- 多すぎはよくない。道をたずねてあらゆる交差点の説明をされたら重要なところも憶えていられない→非協力的

□ Quality: 嘘や mislead を避ける。まさにそれだけ。

□ Relevance: ある情報を提供することによって生じる想像、期待に関連。

- 「プリンタの紙ないんだけど」「角を曲がったとこにキャビネットがあるよ」→キャビネットには紙があることを期待。
- もしキャビネットに紙がなく、相手がそれを知ってたとわかれば、ジョークか意地悪だと思う。

□ Manner: はっきりと、他人が分かるようにものを言うこと。

□ まとめてルール化: 技術製品の協調の原則

□ Don't Impose (マイナスの丁寧さ)

- ユーザの物理的な労力を尊重しよう
- ユーザの心理的な労力を尊重しよう

□ Be Helpful (プラスの丁寧さ)

- 状況に合った適量の情報を早期に提示し、間違いを避けよう (Quantity)
- 問題を解決しよう。不平をたれたり責任転嫁しない (Quantity)
- 予測できるようにふるまおう (Quality)
- 必要な情報だけを要求/提示しよう。mislead しない。(Relevance)
- 平易なことばで説明しよう (Manner)

□ これらのルールは丁寧さのためだけでなく役立つ。

- 対話のメカニズムの代わりに重要なこと〜ユーザが機器を使う理由〜に注目させてくれる
- 例: 3D 空間で複雑な物体を回転→「どうやって回転させるか」よりも「上から見たらどう見えるか」
- 例: ハンドヘルド機器からメッセージを送る→「テキストをどうやって入力するか」よりも「どうやって会話するか」
- 例: 地図上でどこにいるか知らせる→「マップのズームさせかた」より「目的地への行き方」

□ 協調のためのルールに従えば、メカニズムではなくタスクに注目できる。

□ 以下の 3 章では、技術製品に対して協調のルールをどう適用するかについて見ていく。

- ルールをさらに分解してガイドライン化
- さまざまな方面について、正しい例、正しくない例
- これらから、自分の中に「協調的な技術製品」のためのものさしを獲得してほしい。PC でも情報家電でも何でも同じ。
- その後で、実際の開発にそれをどう活かすかの話題に進む。