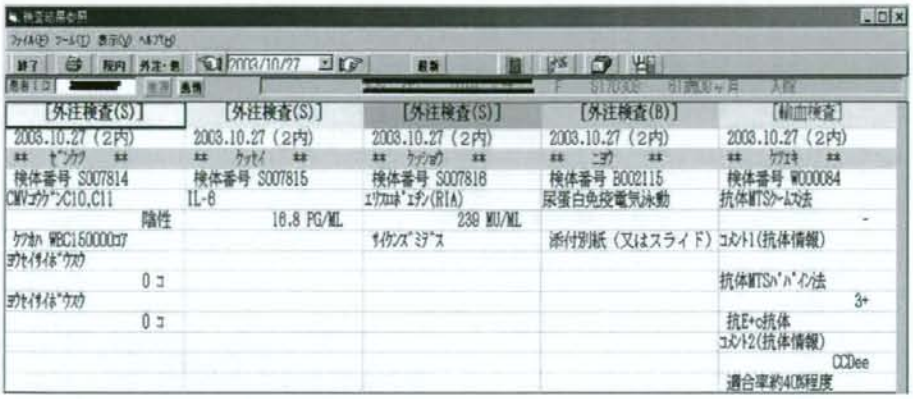
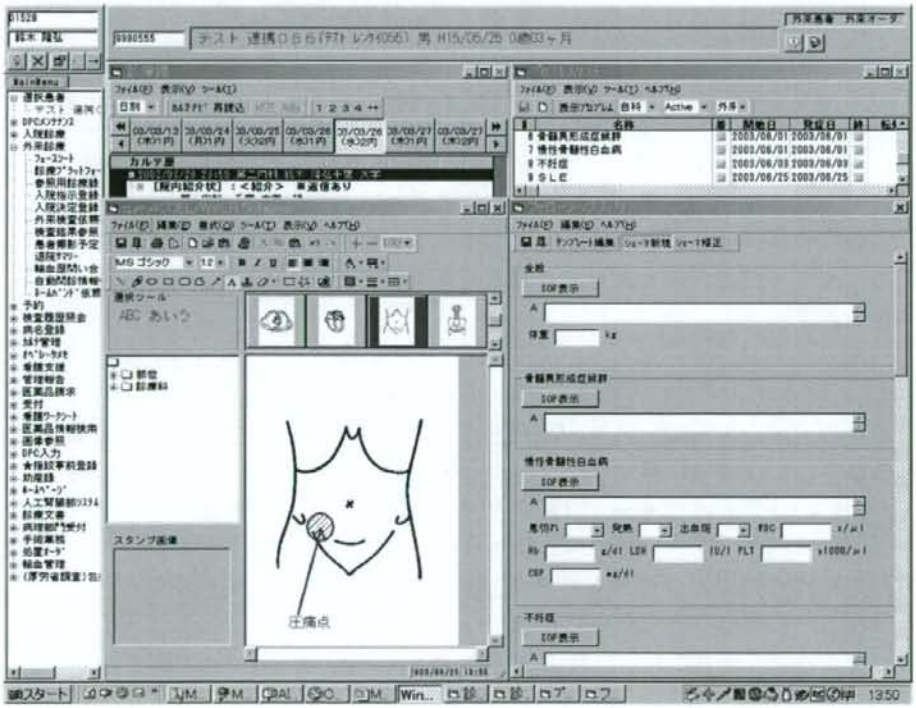


シェーマ記載画面（図 3-2-2）では多数のアイコンを使われて、これらのアイコンは他のシステム（例：Microsoft Word）と同じ意味のアイコンを使っている。

図 3-2-2　シェーマ記載



3）．露出機能という原則（The principle of feature exposure）

ユーザーにできるだけ使える機能をわかるように表示すべきである。

EMR のユーザインターフェースデザインはできるだけ、直覚でわかり易いように画面構成するべきである。例えば、ツールバーやダイアログボックス。

ツールバーやダイアログボックス、ポップアップ（図 3-3-2）、メニュー（図 3-3-3）などは事前に用意したデフォルトが露出される。

テンプレート編集ツール（図 3-3-1）でダイアログボックスなど項目を指定。

図 3-3-1 テンプレート項目設定ツール

図 3-3-2　詳細内容露出

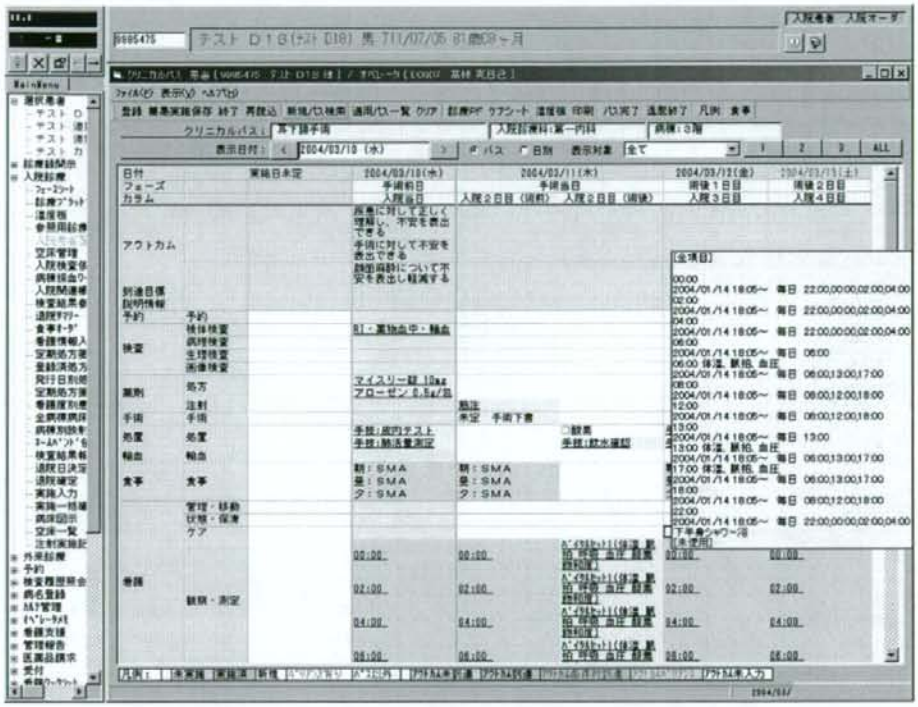
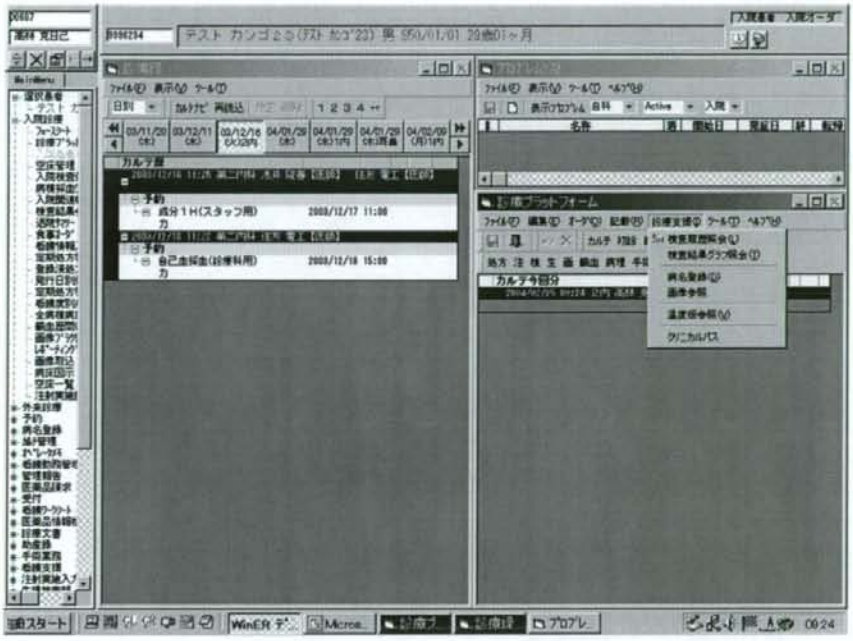


図 3-3-3　メニュー露出



4）．一貫性という原則（The principle of coherence）

プログラムの働きは内部的に外部的に一貫しているべきである。

EMR ユーザインターフェースは標準化され且つ統一されたプラットフォームで作成すべきである。統一されたデザイン、**一貫した方法で、統一された操作方法、すべての画面を標準な構成する。また、用語、**アイコンや色などは一貫性が**必要である。開発された部品を統一管理、再利用する。**これによって、安全性や操作性を向上させられる。

また、通信規格は標準規格の HL7、DICOM3、XML などがよく使われる。

用語について、ＩＣＤ１０は国際死因統計を目標とする分類方法である。以下はそれに関係する標準化分類である。

ＩＣＤ－Ｏ　（Oncology＝病理組織学的分類）

ＩＣＰＭ　（IC of Procedure in Medicine）

ＩＣＤ９－ＣＭ　（ICD9 clinical modification　ＵＳＡ）

ＩＣＰＣ　(IC of Primary Care)
ＩＣＨＣ　(IC of Handicap persons)
ＤＳＭ・ＩＶ　(精神疾患の診断基準)

また、(財) 医療情報システム開発センター(MEDIS-DC)では、標準病名や標準医薬品コード (**表 3-4-1**)など医療情報に関わっている標準化されたマスターや用語が情報提供している。

表 3-4-1　MEDIS-DC 提供されたマスターと用語集一覧


病名マスター(ICD10 対応電子カルテ用標準病名マスター)
(2009.01.01 更新)　new


手術・処置マスター
(2008.08.15 更新)


臨床検査マスター(生理機能検査を含む)
(2008.05.30 更新)


医薬品マスター(HOT番号)


医療機器データーベース


看護実践用語標準マスター
・看護行為編　(2008.12.22 更新)
・看護観察編　(2008.12.22 更新)


症状所見マスター＜身体所見編＞


歯科病名マスター
(2009.01.01 更新)　new


歯科手術・処置マスター


画像検査マスター


J-MIX(電子保存された診療録情報の交換のためのデータ項目セット)

5）．状態の可視化という原則（The principle of state visualization）

システムはユーザーにわかり易いように表現すべきである。

EMR ユーザインターフェースデザインは状態の可視化が重要である。シンプル、グラフィックなどでわかり易く表現する。意味がある絵文字、アイコンなどデザインする。例えば、テキストの検査結果、服薬状況などをグラフ化（図 3-5-2）して、患者に見せながら、病状と治療経過（図 3-5-1）を説明する。

図 3-5-1　熱型表

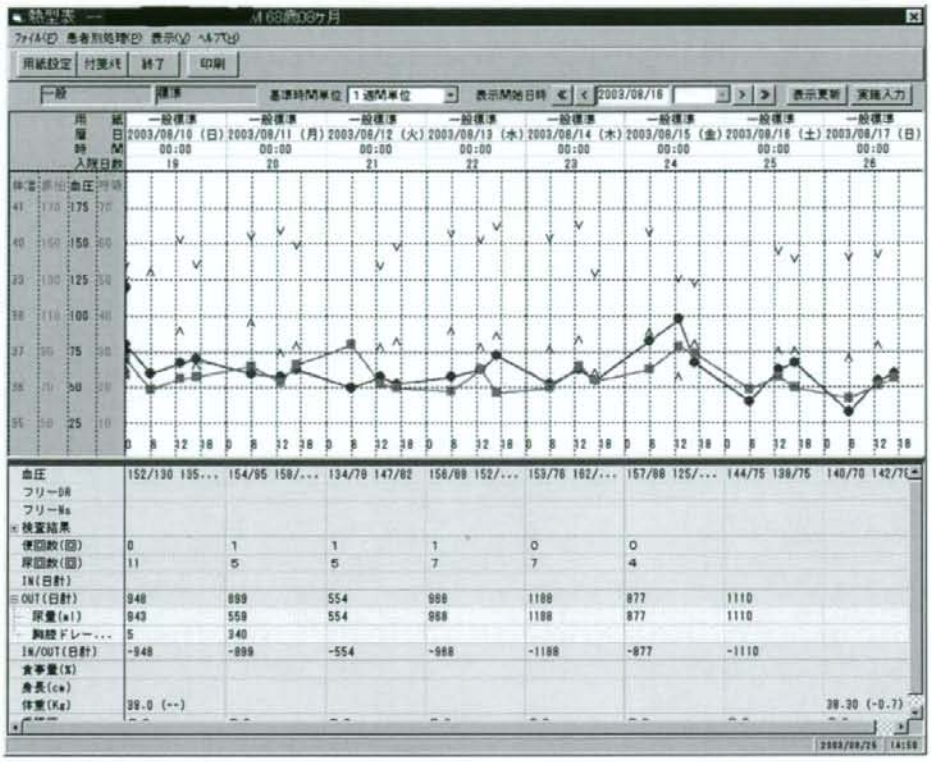
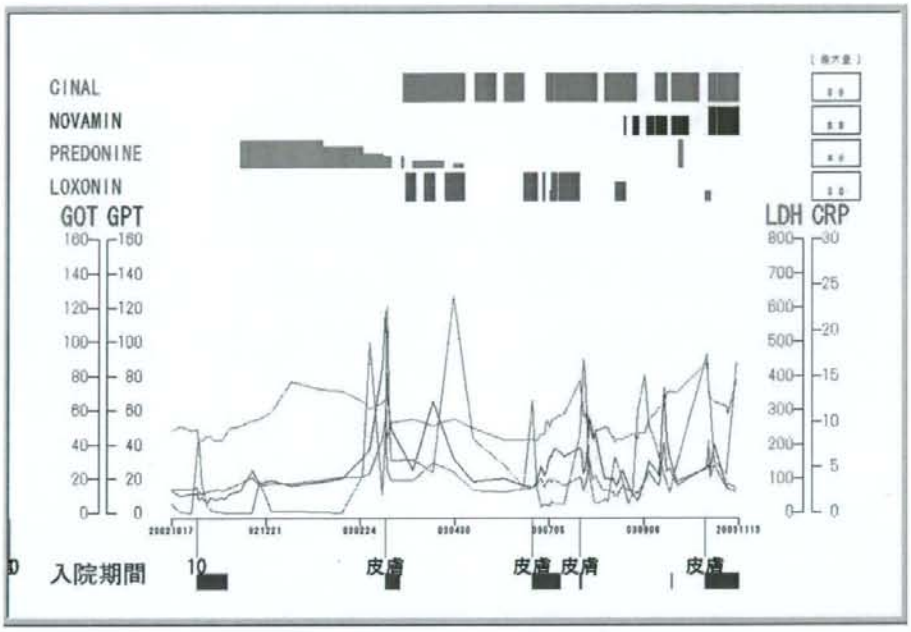

熱型表
ファイル(F) 患者別処理(P) 表示(V) ヘルプ(H)
用紙設定
付箋紙
終了
印刷
一般
標準
基準時間単位
1週間単位
表示開始日時
2003/08/10
表示更新
実施入力
一般標準
2003/08/10 (日)
2003/08/11 (月)
2003/08/12 (火)
2003/08/13 (水)
2003/08/14 (木)
2003/08/15 (金)
2003/08/16 (土)
2003/08/17 (日)
00:00
入院日数
19
20
21
22
23
24
25
26
体温
脈拍
血圧
呼吸
血圧
フリーBR
フリーNs
検査結果
便回数(回)
尿回数(回)
IN(日計)
OUT(日計)
尿量(ml)
胸腔ドレー...
IN/OUT(日計)
食事量(%)
身長(cm)
体重(Kg)
152/130 135...
154/95 158/...
134/70 147/82
156/88 152/...
159/76 102/...
157/88 125/...
144/75 138/76
140/70 142/7
0
1
1
1
0
0
11
5
5
7
7
4
948
899
554
988
1188
877
1110
943
559
554
988
1188
877
1110
5
340
-948
-899
-554
-988
-1188
-877
-1110
39.0 (--)
39.30 (-0.7)
2003/08/26
14:58

図 3-5-2



6）．ショートカットという原則（The principle of shortcuts）

ショートカットは頻繁に使える複雑な操作をより簡便な方法で実現すること。

EMR でのテンプレート（図 3-6-1）やクリニカルパス（図 3-6-2）などはショートカットという原則である。テンプレートでは疾患や病態ごとに、標準的に記載、確認、処置などをユーザインターフェースデザインである。クリニカルパスは病気ごとに整理された指示、処置、観察などをスケジュール軸にあらかじめ登録し、簡単に一括登録及び管理することである。

また、同じようなオーダーパタンは、前回 Do、頻用、セット登録として使うことによって、カルテ入力の手間を大幅に軽減する。

ショートカットキーも頻繁に利用される。

図 3-6-1

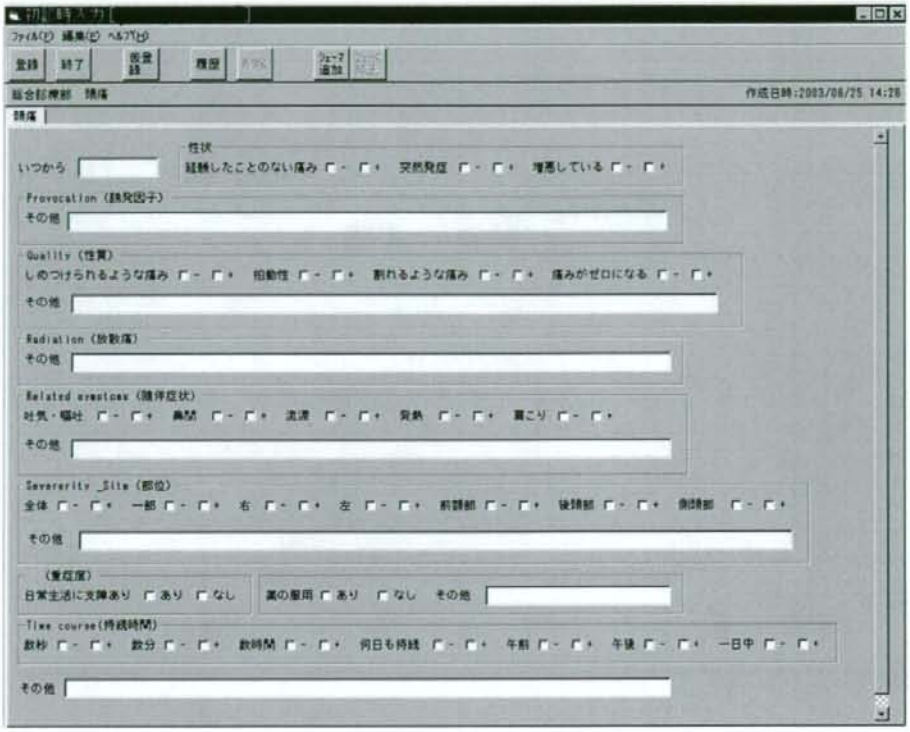
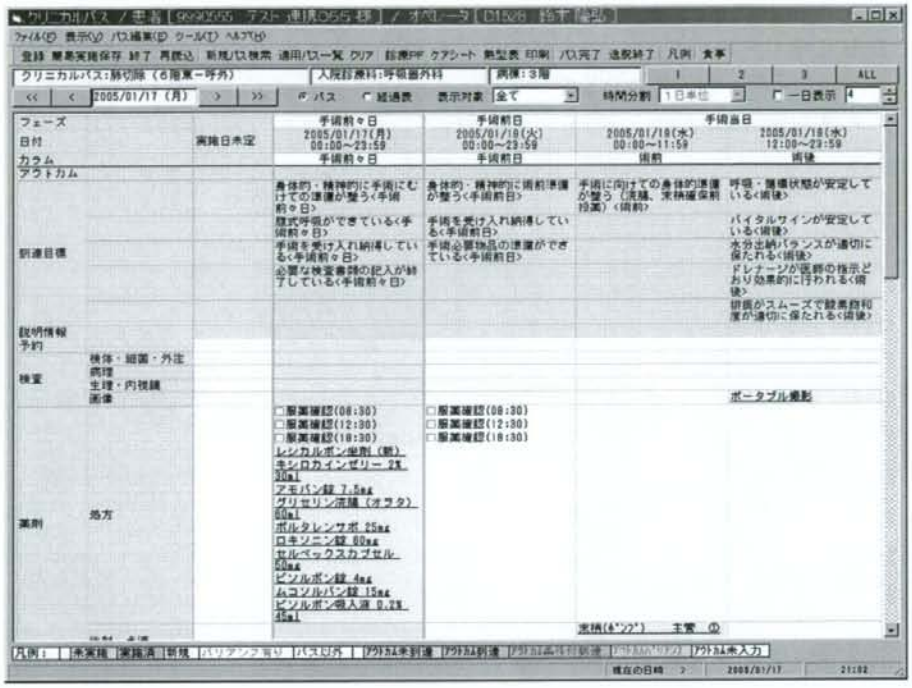


図 3-6-2　クリニカルパス表示画面



7）．　フォーカスという原則（The principle of focus）

フォーカスは人間の注目させること。

EMR のユーザインターフェースデザインではフォーカスが重要である。沢山な情報、記述の中に、異常或いは注意すべきことに対して、字の色や**フォントサイズで表現する。**

アレルギー歴、感染リスクなどを赤字で目立つ場所に表示する。検査結果表示時（図 3-7-1）、正常範囲以下ならブルーで表示し、正常範囲異常なら赤で表示する。

図 3-7-1　検査結果参照画面

8）．文法という原則（The principle of grammar）

ユーザインターフェースは一種の言葉である。

プログラムはコンピュータに実行させる処理の順番を指示するものである。コンピュータが理解するのは０と１の集合であるマシン語であるが、我々が記載するプログラムは高級言語である。高級言語で書かれたプログラムはマシン語に変換して実行ファイルになる。実行処理全体を一括して変換する場合と処理の手順を追いながら変換する場合がある。

プログラムは言語と言われるように文法があり、その文法に沿って命令を書く必要がある。ユーザインターフェースの画面展開なども文法のような手順に従って処理する。デザインには上下関係、主フォームと従フォームの関係などの階層関係を考慮して行う。

9）．ヘルプという原則

ユーザーに必要なヘルプは５種類の基本的なタイプである。

a. 目的指向：「このプログラムによって、どんな事ができるか？」ヘルプと説明書などで、このプログラムはどんなことができるか説明する。

b. 記述的　：「これは何か？ これは何をするか？」よく使われたのはフ

ォームのタイトルにポイントをあわせて、ポップアップで説明すること。ヘルプで言葉の定義やメニュー、操作機能などを解釈する。

c. 手続上　：「どのようにこれをするか？」ヘルプと説明書などで、どのようにこれをするか案内する。

d. 解釈的　：「これはなぜ起こったか？」エラーが発生するときのメッセージとして、よく使われる。

e. ナビゲーション：「ここはどこか？」適切な全体のインターフェイス仕様により回答することができる。

EMR のユーザインターフェースのデザインはこの５種類のヘルプが必要である。

１０）. 安全性という原則（The principle of safety）

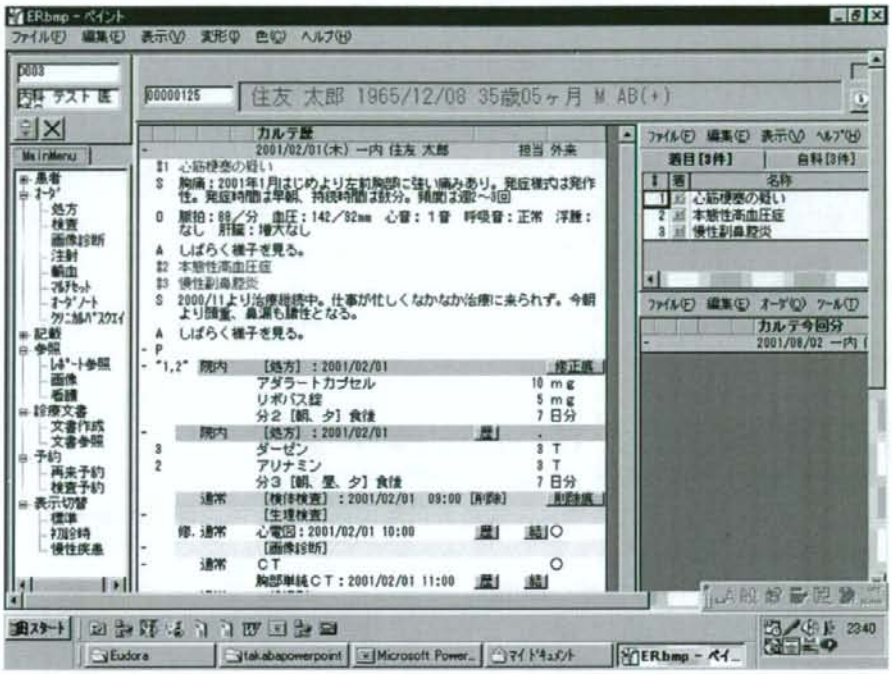
安全を信頼すべきものとし、リスク最小限にするべきである。

誰でも何時でも何処でも安全・安心の医療が必要となる。患者の安全は第一と考えている。

EMR では情報の共有、医療の標準化、重複検査・薬剤の重複処方の防止、入院患者にリストバンド（図 3-10-1）を使用し、認証によって、取り違いを防止し、薬物の投与量（年齢、体重、性別）の監査、用法の監査、薬剤禁忌の監査、薬剤相互作用等のチェック（図 3-10-1）、実施にバーコードによる患者認証を行い、点滴注射の間違い防止、クリニカルパスなどさまざまの方法を利用し、事故防止を図る。

図 3-10-1　リストバンド

図 3-10-1　処方情報



１１）．コンテキストの原理（The principle of context）

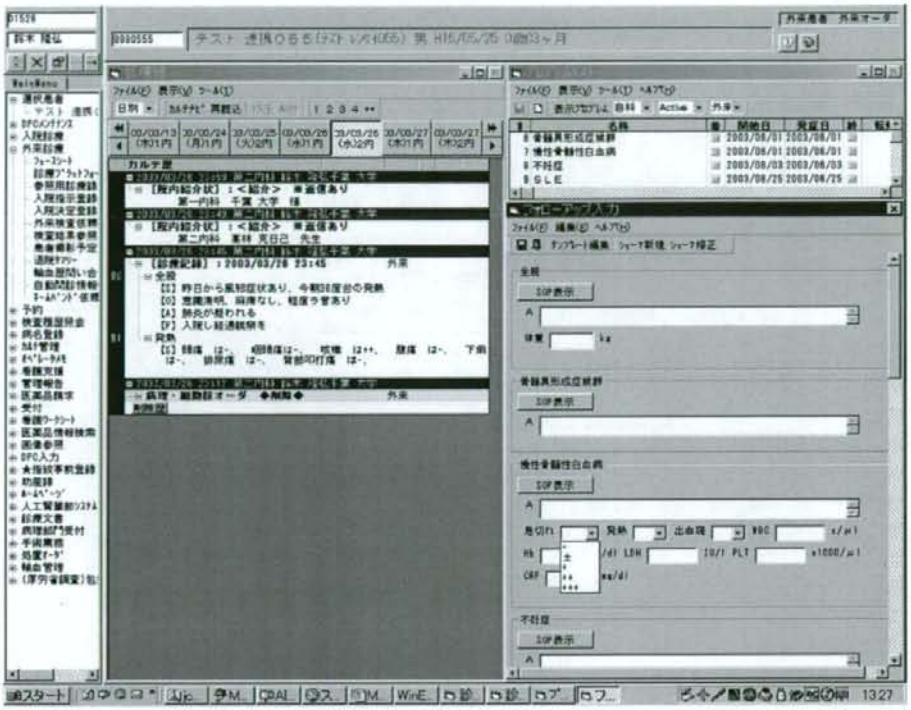
ユーザー活動を１つの明確なコンテキストに制限すべき。

個々のユーザー行動は、与えられたコンテキスト内で起こる。例：コンボボックス（図 3-11-2）、ダイアログボックス、チェックボックス、リストボックスなど。

EMR では、テンプレート作成ツールにより、各診療科独自の電子カルテ画面を作成すること。テンプレート（図 3-11-1）は入力ボックスや選択ボックス、チェックボックス等の様々な入力形式があるが、構造化されたテキストオブジェクトを選ぶこともできる。即ち、事前に用意された標準化された定型用語の入力パターンを選択すること。

図 3-11-1　テンプレート入力画面

図 3-11-2

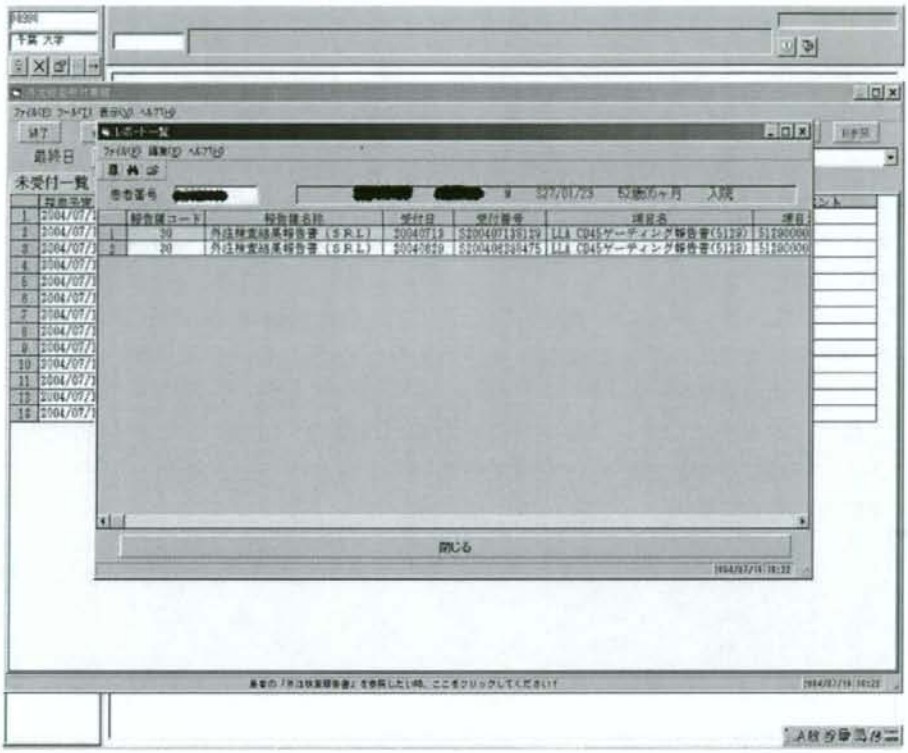


１２）．美学という原則（The principle of aesthetics）

画面のデザイン、ボタンの配置など、シンプルかつわかり易いように表現すべきである。

EMR の画面デザインは芸術的な美しさまで要求されると思わないが、シンプルなデザイン（図 3-12-1）、わかり易く且つ操作性がよくて、スピードが速いことを前提に、美学的なデザインのほうが好ましい。

図 3-12-1



１３）．ユーザーテストという原則（The principle of user testing）

ユーザーインタフェーステストはすなわち実際のエンドユーザーを使って、テストを行う。これはユーザインターフェースデザインの欠陥を発見、検出するために非常に有効な手法である。また、ユーザーテストの過程を観察し、バグやわからない点を改善することができる。

EMR ではエンドユーザーのテストや確認により、欠陥の発見のみならず、業務流れの改善、操作性の向上など貴重な意見も得られて、ユーザインターフェースデザインに対して、果たす役割は極めて大きいである。

１４）．謙虚さという原則（The principle of humility）

謙虚は普通の人々が言いたいことを聴くこと。他人の意見を謙虚に聞き、欠陥を見分ける能力を増大させる。理想は多くのユーザーの意見を聞いて、さらに開発者としての洞察を持って、デザインすること。

EMR では時間をかけて、謙虚にさまざまな意見を取り入れて、徐々に作り上げたものである。

4．まとめ

電子カルテのユーザインターフェースデザインは人が安全にかつ安心にシステムを利用できるための重要部分である。ここでは、Talin氏のコンピュータシステムにおけるユーザインターフェースデザインのために１４項目の基本的な原則を紹介し、EMRのユーザインターフェースデザインには１４項目の基本的な原則が適応と思われる。

また、電子カルテでは一貫性のみならず、標準化も重要であり、なお安心・安全の提供、セキュリティの確保、生産性の向上などのユーザインターフェースとして求められる。

今後、ユーザインターフェースに対して、利便性・信頼性の向上が不可欠であり、それに伴い、直感的に利用できるユーザインターフェースのデザインが重要となるのである。

参考文献

[1] A Summary of Principles for User-Interface Design.

by Talin

This document represents a compilation of fundamental principles for designing user interfaces, which have been drawn from various books on interface design, as well as my own experience. Most of these principles can be applied to either command-line or graphical environments. I welcome suggestions for changes and additions — I would like this to be viewed as an "open-source" evolving document.

1. The principle of user profiling

— Know who your user is.

Before we can answer the question "How do we make our user-interfaces better", we must first answer the question: Better for *whom*? A design that is better for a technically skilled user might not be better for a non-technical businessman or an artist.

One way around this problem is to create user models. [TOG91] has an excellent chapter on brainstorming towards creating "profiles" of possible users. The result of this process is a detailed description of one or more "average" users, with specific details such as:

- What are the user's goals?
- What are the user's skills and experience?
- What are the user's needs?

Armed with this information, we can then proceed to answer the question: How do we leverage the user's strengths and create an interface that helps them achieve their goals?

In the case of a large general-purpose piece of software such as an operating system, there may be many different kinds of potential users. In this case it may be more useful to come up with a list of *user dichotomies*, such as "skilled vs. unskilled",

"young vs. old", etc., or some other means of specifying a continuum or collection of user types.

Another way of answering this question is to talk to some real users. Direct contact between end-users and developers has often radically transformed the development process.

2. The principle of metaphor

— Borrow behaviors from systems familiar to your users.

Frequently a complex software system can be understood more easily if the user interface is depicted in a way that resembles some commonplace system. The ubiquitous "Desktop metaphor" is an overused and trite example. Another is the *tape deck* metaphor seen on many audio and video player programs. In addition to the standard transport controls (play, rewind, etc.), the tape deck metaphor can be extended in ways that are quite natural, with functions such as *time-counters* and *cueing buttons*. This concept of "extendibility" is what distinguishes a powerful metaphor from a weak one.

There are several factors to consider when using a metaphor:

- Once a metaphor is chosen, it should be spread widely throughout the interface, rather than used once at a specific point. Even better would be to use the same metaphor spread over several applications (the tape transport controls described above is a good example.) Don't bother thinking up a metaphor which is only going to apply to a single button.
- There's no reason why an application cannot incorporate several different metaphors, as long as they don't clash. Music sequencers, for example, often incorporate both "tape transport" and "sheet music" metaphors.
- Metaphor isn't always necessary. In many cases the natural function of the software itself is easier to comprehend than any real-world analog of it. Don't strain a metaphor in adapting it to the program's real function. Nor should you strain the meaning of a particular program feature in order to adapt it to a metaphor.
- Incorporating a metaphor is not without certain risks. In particular, whenever physical objects are represented in a computer system, we inherit not only the beneficial functions of those objects but also the detrimental aspects.

- Be aware that some metaphors don't cross cultural boundaries well. For example, Americans would instantly recognize the common U.S. Mailbox (with a rounded top, a flat bottom, and a little red flag on the side), but there are no mailboxes of this style in Europe.

3. The principle of feature exposure

— Let the user see clearly what functions are available

Software developers tend to have little difficulty keeping large, complex mental models in their heads. But not everyone prefers to "live in their heads" — instead, they prefer to concentrate on analyzing the sensory details of the environment, rather than spending large amounts of time refining and perfecting abstract models. Both type of personality (labeled "Intuitive" and "Sensible" in the Myers–Briggs personality classification) can be equally intelligent, but focus on different aspects of life. It is to be noted that according to some psychological studies "Senseless" outnumber "Intuitive" in the general population by about three to one.

Intuitive prefer user interfaces that utilize the power of abstract models — command lines, scripts, plug–ins, macros, etc. Senseless prefer user interfaces that utilize their perceptual abilities — in other words, they like interfaces where the features are "up front" and "in their face". Toolbars and dialog boxes are an example of interfaces that are pleasing to this personality type.

This doesn't mean that you have to make everything a GUI. What it does mean, for both GUI and command line programs, is that the features of the program need to be *easily exposed* so that a quick visual scan can determine what the program actually does. In some cases, such as a toolbar, the program features are exposed by default. In other cases, such as a printer configuration dialog, the exposures of the underlying printer state (i.e. the buttons and controls which depict the *conceptual printing model*) are contained in a dialog box which is brought up by a user action (a feature which is itself exposed in a menu).

Of course, there may be cases where you don't wish to expose a feature right away, because you don't want to overwhelm the beginning user with too much detail. In this case, it is best to structure the application like the layers of an onion, where peeling