Main Goal of "Bridging the Gap"

The main goal of this application design guide is to bridge the gap between analysis and implementation. Let us have a closer look at what we mean by this!

Analysis

Collecting data on users, tasks and the work environment has become a prominent part of the SAP software development process. Several analytical methods have been developed or adopted from outside sources and are rolled out to the application developers and user interface designers: Brainstorming sessions, site visits, reviews, user days, and design sessions are the most important ones (you find these methods in the "Analysis" section of the Design Guild). All these methods supply developers with a wealth of data. But how do these data find their way into the application and interface design? Which aspects of them are used and how?

Design by Implementation?

In their daily work developers are confronted with the basic building blocks for user interfaces, as there are screens and controls. There seems to be no or only a loose connection between the data from the analyses and the screen elements. Thus, interface design is often regarded as and reduced to the task of putting controls on screens and arranging them in a more or less usable and pretty fashion. This view, however, falls far too short, because it disregards the users and their needs as well as the context of the task.

So, how can developers connect the results of the analyses to the task of designing screens and using controls? For long this question had a simple answer: No real connection was made! There were no analyses carried out, and interface design, for which little time was left anyway, often happened in a "last minute" fashion. Again, such an approach is in danger not to lead to usable applications that come up to the users' needs and expectations.

This is changing now! Developers are willing to do those analyses and let users participate in the development process. But there still remain some puzzles about how the results of the analyses can be utilized for the design process.

Bridging the Gap

What is needed - and that is what we want to provide here (see Part I: Design Approach) - are decision criteria, guidelines, or just rules of thumb, simple and practical methods as well as design options and background knowledge for application and interface design that are based on the results of the above-mentioned analyses, and thus bridge the gap between analysis and implementation.

The rules and criteria in this design guide cover aspects like structuring applications, structuring screens and presenting data and functionality based on requirements made by users, tasks and processed data.

In the end, there should be a new user-oriented view of the task and of the cooperation between user and system.

Note however, that these design guidelines do not replace the SAP Style Guide (in the "Implementation" section of the Design Guild you find the condensed version of the SAP Style Guide with new additions)! These guidelines cover the general and more abstract issues of application design that characterize the first design stages. The SAP Style Guide comes into play as soon as you begin with the fine-grained screen and menu design.

Source: From Analysis to Design: Bridging the Gap
Main Goal of "Bridging the Gap"

Analysis | Design by Implementation? | Bridging the Gap

The main goal of this application design guide is to bridge the gap between analysis and implementation. Let us have a closer look at what we mean by this!

Analysis

Collecting data on users, tasks and the work environment has become a prominent part of the SAP software development process. Several analytical methods have been developed or adopted from outside sources and are rolled out to the application developers and user interface designers: Brainstorming sessions, site visits, reviews, user days, and design sessions are the most important ones (you find these methods in the "Analysis" section of the Design Guild). All these methods supply developers with a wealth of data. But how do these data find their way into the application and interface design? Which aspects of them are used and how?

Design by Implementation?

In their daily work developers are confronted with the basic building blocks for user interfaces, as there are screens and controls. There seems to be no or only a loose connection between the data from the analyses and the screen elements. Thus, interface design is often regarded as and reduced to the task of putting controls on screens and arranging them in a more or less usable and pretty fashion. This view, however, falls far too short, because it disregards the users and their needs as well as the context of the task.

So, how can developers connect the results of the analyses to the task of designing screens and using controls? For long this question had a simple answer: No real connection was made! There were no analyses carried out, and interface design, for which little time was left anyway, often happened in a "last minute" fashion. Again, such an approach is in danger not to lead to usable applications that come up to the users' needs and expectations.

This is changing now! Developers are willing to do those analyses and let users participate in the development process. But there still remain some puzzles about how the results of the analyses can be utilized for the design process.

Bridging the Gap

What is needed - and that is what we want to provide here (see Part I: Design Approach) - are

- decision criteria, guidelines, or just rules of thumb,
- simple and practical methods as well as
- design options and background knowledge

for application and interface design that are based on the results of the above-mentioned analyses, and thus bridge the gap between analysis and implementation.
The rules and criteria in this design guide cover aspects like structuring applications, structuring screens and presenting data and functionality based on requirements made by users, tasks and processed data.

In the end, there should be a new user-oriented view of the task and of the cooperation between user and system.

Note however, that these design guidelines do not replace the SAP Style Guides! These guidelines cover the general and more abstract issues of application design that characterize the first design stages. The SAP Style Guides comes into play as soon as you begin with the fine-grained screen and menu design.

Source: From Analysis to Design: Bridging the Gap
Why a Design Guide? What Do You Find Here?

Why this Guide?

As we already stated, the main goal of this application design guide is to bridge the gap between analysis and implementation. To our opinion this gap exists, because developers often do not have sufficient knowledge for successfully applying the results from the analysis phase to the implemented design. They need to know which "parameters" - based on the data collected in the analysis phase - determine the design of applications. As we feel, application design is still considered as an art - often, the design pops up like "magic". Even thick textbooks on design are not very specific with respect to the criteria that are relevant for application design. This guide is an attempt to shed some light on the mystery.

For more details on our approach see Elements of Application Design!

What Do Developers Really Need - Examples Only?

Sure, 99% of the current software is written without a design guide and by looking at examples. Examples are a very efficient source of inspiration for application design. So, one might argue that some good examples is all developers need. But is this right? Examples have their limitations: They are implicit and they are prototypical.

With "implicit" we mean that if you follow an example, you will do it right "automatically". That's OK in many cases, but often you do not know why you did it right and which rationale and principles lead to the specific design.

This is even more of a problem, if your application does not fit the examples, because these are "prototypical " and cover just the basic cases. But if the examples break down, you're on your own. You have to learn for yourself which dimensions are are relevant for application design.

In this situation, this guide comes in handy - as we hope. It provides a simple design methodology that focuses on "usable" criteria that directly lead to design decisions.

Audience

This design guide is primarily intended for usability people and interface designers, that is people who consult application developers or work together with them in teams. We know that application developers themselves usually want more "hands on" and immediately usable guidelines. Therefore, we recommend this guide only for those application developers who want to gain more insights and background knowledge on application design issues.

In spite of this, we use the term "developer" for the reader, because in the end we are aiming at the developers, although we may reach them only indirectly through the mediation of usability people and interface designers.

Domain
Why a Design Guide?

While the more general aspects of this design guide apply to any kind of software, the more specific parts like interaction patterns, application patterns, screen types etc. are tailored to data processing applications in the business domain. We do not cover applications like drawing or painting programs, text processing and presentation software, or games.

Overview

In the course of this guide, we point out general design principles, dimensions and criteria and show how they determine the resulting design. In addition, we present overview tables on some of the design dimensions and even provide a few quick & dirty methods which may help you to answer some of the questions that arise during the design process.

Part I: Design Approach

In Part I we present our approach to application design. Here we show how the results from task analysis and user analysis, typically in the form of a scenario, find their way into the design process. Our approach is based on principles, dimensions, criteria and classifications; the latter serve as background knowledge. The main parameters for design decisions are the criteria and their values. We also present an overview of the criteria used in this guide.

Part II: Design Focus

Part II of this guide is meant to create a "mindset" for developers: First we present the focus issues for application design: task, users, processing and display needs, application structure and navigation, presentation, and last, but not least, user support. In addition, we propose a number of user-oriented design principles. These principles not only sharpen your eyes for the users' need and abilities, but they already lead to practical conclusions about user-oriented application design.

Part III: Defining the Design

Part III is the "core" of our design guide. Here we utilize the results from prior development phases, like task analysis and user analysis, as design criteria.

First we outline our proposal for a simple design methodology bases on the elements principles, properties, dimensions and criteria, design options, and classifications. Then we look for design parameters for tasks, data, application structure and navigation. We also look at the users and their work place and at the presentation of data and functionality.

We conclude this part with an outlook on the next design steps as well have a look at why designs may fail and what are the stumbling blocks for good interface design.

Part IV: Background Knowledge

In Part IV we include additional information that may be useful for getting a broader understanding of the design process. This is the "classification" part of our approach.

What is Missing?

This design guide covers only the more abstract aspects of application design. It does not cover the issues of how controls are selected and arranged, the details of screen and menu design, terminology, etc. It does also not cover prototyping and user testing. See the respective sections and the compact version of the SAP Style Guide in the Design Guild for more information on these topics!
Why a Design Guide?

Source: From Analysis to Design: Bridging the Gap
Elements of Application Design

What is What? | The Top-Down vs. Bottom-Up Dilemma | From Scenario(s) to Application Structure and Navigation: Defining the High-Level Structure | Tasks and Data (Processes and Objects) | Users and Work Place | Summary

In Main Goal of "Bridging the Gap" we asked how the data from the analyses find their way into the application and interface design and which aspects of them are used and how. This question has found different answers by various authors. To name a few examples, Hugh Beyer and Karen Holtzblatt developed the Contextual Design methodology, Larry Constantine proposes Essential Use Cases, and Alan Cooper promotes Goal-directed Design.

In this design guide we present an approach that complements these methods; it focuses on

- principles
- properties
- dimensions and criteria
- design options
- classifications

that are relevant for design decisions. That is, we agree to the basic approaches of these authors, but especially with providing criteria for design decisions and background knowledge we try to remove a little bit of the mystery that still surrounds the final steps on the way from analytical data to design decisions.

We complement these elements by some simple practical methods and overview tables.

What is What?

Principles

Principles are general rules that guide the design. Such a principle might simply state "Make it simple!". In User-Oriented Design Principles we propose design principles that take care of the users, their goals and needs. Currently we do not propose further principles.

Properties, Dimensions and Criteria

Properties are characteristics of the application design (or the final application) itself. For example, an application design has a certain structure and navigation. It also implements certain processes in order to accomplish a task. It further processes certain data. We consider presentation also as a design property.

Dimensions and criteria determine how the properties will be realized. That is, dimensions and criteria are parameters or boundary conditions that restrict the possible design space with respect to certain properties.

Dimensions are the general aspects of the parameters. A dimensions may have several criteria which each may have possible values. Such criteria might even overlap - usefulness is what counts, not "formal" considerations. As an example, the dimension users has the criteria computer knowledge and domain knowledge. Each criterion, that is, type of knowledge, may take on different values like Beginner, Intermediate, and Professional. Note that though there may be a continuum of values, for practical reasons it makes more sense to consider only a few typical values which are useful for making design decisions.
**Design Options and Classifications**

*Design options* are interface elements, interface patterns, screen divisions, application structures or even more abstract entities that can be used in user interface designs. The dimensions, criteria, and their values determine which of these options are relevant for a design.

*Classifications* move the design options to a more abstract level. They provide *prototypical cases* for certain abstract or concrete design elements like tasks, data structures, screens, applications, or application structures. They help developers to abstract their designs into the direction of "prototypical" cases. Matching to these cases allows them to utilize proven design solutions as well as to learn the characteristics, advantages and disadvantage of these solutions. Thus, classifications are useful in the daily work and also help developers to extend their design knowledge.

In the introduction we discussed *examples* as an often-used design aid. We stated there that examples may be a valuable source of inspiration for application designers, but that they also have their limitations. Classifications are more general in scope and may be extended as new developments come up.

In the following, we shortly outline the basic process of feeding data into the design, and especially how *properties* and *dimensions* come into play.

**The Top-Down vs. Bottom-Up Dilemma**

The top-down vs. bottom-up dilemma emphasizes the different approaches to interface design. If you start from the data gathered during analysis you are using the *top-down approach*: Usually there are scenarios, which are like stories. They describe more or less detailed task steps, but the connection to a "real" user interface is very lose and vague. On the other hand, if you start with arranging screen elements into functional units and assembling these into whole screens or screen sequences, you are moving *bottom-up*. Though very different, both of these approaches have the same goal, namely to create a user interface that enables users to fulfill a certain task - as described in the scenario or a formal specification.

A pure bottom-up process does not make much sense, because it does not structure the design process, especially with respect to more global aspects of the design. You cannot design a screen by putting some fields on a blank template without an understanding of the global structure of the screen, the task flow and dependencies between screen elements. Thus, you - hopefully - will always have some sort of global sketch - at least in mind - when you proceed with screen design.

A pure top-down approach, on the other hand, may be too abstract for many developers - it takes too long, before they really get their "hands on" the interface design. It also does not take enough care of the dependencies between actual screen elements and interaction.

So, how can these two approaches be combined? We propose to go top-down first and then move bottom-up. In reality things will be even more intermingled, that is, you will repeatedly switch between the more global top-down view and the more local bottom-up view. The top-down approach serves as a means for organizing the design process, while the design itself is a mixture of bottom-up and top-down steps.

**From Scenario(s) to Application Structure and Navigation: Defining the High-Level Structure**

In agreement with the above-mentioned authors we base the design phase on a *scenario* (or a set of scenarios) which includes:

- one or more *task descriptions*,
- a description of the *typical users*, and
Elements of Application Design

- an optional description of the work environment and the communication structures therein.

This scenario may exist as a story with "real" persons like Coopers "personas", it may be an elaborate set of models, visions and story boards, as the Holtzblatt methods provides, or it may only be a short list of task steps and a user role created during a brainstorming session. You can, of course, debate which of these methods is the best one, but we do not want to take position for any of these approaches - you will agree that any of these is much better than none. All of these methods are useful, and which suits your purposes best may also depend on your personal preferences. In addition, factors like time pressure, opportunity to carry out site visits, as well as the circumstances under which the data were collected influence which kind of data are at your disposal.

The first step is to use the scenario to create a high-level, abstract structure of the task: The scenario is broken down into one or more sequences of steps which accomplish the task. For more complex tasks the scenario may have to be divided into several linked subscenarios with subtasks. Alternatively, there may already exist a number of scenarios that describe the activities for a work place and that have to be connected. Thus, breaking down the scenario determines the application structure, while flow of control has influences on structure and navigation. Depending on the complexity of the task and the resulting application, the emerging structure may be very abstract in the beginning and has to be refined iteratively. Thus, we follow a typical top-down approach. In other cases the application may be rather simple and its structure and navigation may be self-evident and not a real "design issue".

The structural units that emerge in this phase, Holtzblatt calls them "focus areas", typically serve a single purpose and, on the interface side, will be represented by screens, pages or larger areas within these. In more complex cases, a screen or page will not suffice, and these units have to be distributed over more screens, thus increasing the need for navigation. As structural units serve a certain purpose, or sometimes several ones, they contain a certain functionality for processing the required data. If several units are necessary for accomplishing the task, these are linked through navigational paths. We recommend to draw a sketch of the structure and the links either on paper or using graphic tools.

This approach is similar in spirit to the user environment design (UED) by Hugh Beyer and Karen Holtzblatt, which is a more developed and more formal methodology. See their book for more details!

We discuss structure and navigation issues in Structure and Navigation - General Considerations and Structure and Navigation - How to Optimize It.

Tasks and Data (Processes and Objects)

Users work in a structural unit like a focus area to fulfill a task or subtask. This task or - in a more abstract view - process can usually be broken down into one or more sequences of steps, maybe with alternative paths. These steps correspond to simpler tasks. We call them "task patterns", if they are found in many applications. Task patterns may also be composed of even simpler steps which we call "elementary tasks" (see below).

The above-mentioned steps should already be defined in the scenario(s). Often, however, scenarios are ill defined or incomplete, and you have to make them more precise. Design sessions provide a good opportunity to fix scenarios (see the Design Guild for details on Design Sessions!).

The scenario should also provide sufficient information about the data (objects) that are being processed by the task. Of course, the data need not be known in detail, but the relevant data and their properties (see Data and their Requirements) should be extractable from the scenario.

Mapping Task Steps to Interface Elements

We propose to strip off the "meaning" of the task steps, that is, to identify their abstract contents, and to map them to abstract task steps which we call "elementary tasks". Such an abstraction makes it easier to find a mapping to interface elements,
interface patterns, or even proven interface solutions. We use simple "interaction patterns" as a link between tasks and interface elements. The mapping is, however, usually not a one-to-one mapping, because processes act on data, and data impose certain requirements that have to be taken into account when choosing designs.

**Time and Screen Space**

Many of the design decisions are influenced by temporal and screen space requirements. Therefore, we propose the two dimensions **time** (temporal requirements) and **space** (screen space requirements). See Tasks and Their Requirements for details!

**Users and Work Place**

Last, but not least, **users** and **work place** also act as "restrictions" for design decisions.

Information on users and work places should also be included in the scenarios. **Users** may be described in a more abstract fashion as "prototypical users", user groups, populations, etc., or in a more concrete fashion like the "**personas**" (primary, secondary) promoted by Alan Cooper. Irrespective how users are described, the scenario should provide information about the users’ computer and domain knowledge as well as about personal goals, needs and preferences.

**Work place** information is often very scarce in business scenarios. However, as Beyer and Holtzblatt point out, the working environment may be a valuable source of information for improving business processes. Often the developers' perspective is limited to their application and disregards working environment, communication structure and company culture. These factors may in reality have a much bigger impact on effectively than the software itself.

See Users and Work Places for criteria and details!

**Presentation**

**Presentation** refers to the how way objects on the screen are presented (or hidden...). These objects may be data, functions, options, context information, instructions etc. Usually a scenario tells little or nothing about presentation. How objects are presented on the screen depends very much on other dimensions like time and space requirements, users and work places.

We distinguish between the dimensions **data** and **functionality**. There may be other distinctions, like options, but we restrict our discussion - which is on a very abstract level - to the two. For the criteria see the respective sections!

You will also find references to the presentation in Users and Work Places. For more detailed information on presentation issues see the sections on **visual design** in the Design Guild.

**Design Options and Classifications**

**Design options** are not part of the specifications or scenarios, but are part of the design knowledge of usability people, user interface designers and developers. The goal of this guide is to present parameters for selecting the right design options.

**Classifications** are usually used only implicitly in application design. Developers have a "natural" understanding of how things
are structured. This knowledge is especially well developed for data; however, we assume that in other areas those ideas are relatively vague and based on examples and experience. For this reason we provide a couple of classifications - many originate from the work with the R/3 system - in order to provide developers with a firmer foundation in other application aspects too.

We collected the classifications in Part IV: Background Knowledge as a reference source. This way we could keep Part III: Defining the Design shorter. When suitable we link to the respective section in the reference part.

### Summary

<table>
<thead>
<tr>
<th>Design Element</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Principles</strong></td>
<td></td>
</tr>
<tr>
<td>● User-oriented design principles</td>
<td></td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td></td>
</tr>
<tr>
<td>● Structure and navigation</td>
<td></td>
</tr>
<tr>
<td>● Tasks (Processes)</td>
<td></td>
</tr>
<tr>
<td>● Data (Objects)</td>
<td></td>
</tr>
<tr>
<td>● Presentation</td>
<td></td>
</tr>
<tr>
<td><strong>Dimensions</strong></td>
<td></td>
</tr>
<tr>
<td>● Time = temporal requirements (for tasks and data)</td>
<td></td>
</tr>
<tr>
<td>● Space = screen space requirements (for tasks and data)</td>
<td></td>
</tr>
<tr>
<td>● Users (for tasks, data, presentation)</td>
<td></td>
</tr>
<tr>
<td>● Work place (for tasks, data, presentation)</td>
<td></td>
</tr>
<tr>
<td>● Data, functionality (for presentation)</td>
<td></td>
</tr>
<tr>
<td><strong>Classifications</strong></td>
<td></td>
</tr>
<tr>
<td>● Elementary tasks</td>
<td></td>
</tr>
<tr>
<td>● Interface patterns</td>
<td></td>
</tr>
<tr>
<td>● Screen types</td>
<td></td>
</tr>
<tr>
<td>● Application patterns</td>
<td></td>
</tr>
<tr>
<td>● Applications structures and metastructures</td>
<td></td>
</tr>
</tbody>
</table>

As dimensions depend on properties, we added relevant properties to the dimensions. In Part III: Defining the Design we further elaborate the dimensions by introducing criteria that can take on values; these act as "boundary conditions" and restrict the space of the design options. See also Overview of the Design Parameters for a reference of all parameters discussed in this guide. You can use this overview for creating a "design checklist".

*Part II: Design Focus* presents user-oriented design principles. In *Part III: Defining the Design* we have a look at dimensions, criteria and values for the properties task, data, structure and navigation. User and work place impact on design and presentation are covered in Users and Work Places. We also discuss presentation of data and of functionality at the end of part III. The classifications are contained in *Part IV: Background Information*. 
Selected References


Alan Cooper (1999). *The inmates are running the asylum*. Indianapolis, IN: SAMS.

Source: From Analysis to Design: Bridging the Gap
Overview of the Design Parameters

Task (Processes) | Data (Objects) | Structure and Navigation | Presentation

The following overview of properties, dimensions and criteria, and - where existing - design options can be used as a reference and as basis for a checklist for design issues. For details see the respective sections (links are found below the tables)!

### Task (Processes)

#### Dimension: General Task Characteristics

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus</td>
<td>Editing - Inspecting - Browsing</td>
</tr>
<tr>
<td>Objects</td>
<td>One (Simple or Complex) - Many Simple - Many Complex</td>
</tr>
<tr>
<td>User Support</td>
<td>None - Basic - Extensive</td>
</tr>
</tbody>
</table>

#### Dimension: Space (Processing and Display Requirements)

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing and Display Needs: One vs. Several Items</td>
<td>Access to Multiple Objects/Object Parts Necessary - Access to Single Access to Objects/Object Parts Possible</td>
</tr>
<tr>
<td>Processing and Display Needs: Overview - Detail</td>
<td>Needed in Parallel - Not Needed in Parallel - Not Applicable</td>
</tr>
<tr>
<td>Data Objects</td>
<td>Single Attributes - Multiple Attributes for One Object - Single Attribute for Many Objects - Multiple Attributes for Many Objects</td>
</tr>
<tr>
<td>Presentation</td>
<td>Needs Little Space - Needs Much Space</td>
</tr>
<tr>
<td>Support Information</td>
<td>None - Yes, Needs Space</td>
</tr>
</tbody>
</table>
### Overview of the Design Parameters

#### Dimension: *Time (Processing and Display Requirements)*

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel Access to Objects or Object Parts:</td>
<td>Never - Temporary - Permanent</td>
</tr>
<tr>
<td>Duration</td>
<td></td>
</tr>
<tr>
<td>Processing Time</td>
<td>Irrelevant - High Speed/Volume Processing</td>
</tr>
</tbody>
</table>

**References:** [Tasks and Their Requirements, Gathering Requirements for Tasks]

#### Dimension: *Users*

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Knowledge</td>
<td>Beginner - Intermediate - Proficient</td>
</tr>
<tr>
<td>Domain Knowledge</td>
<td>Little or none - Basic - Good</td>
</tr>
<tr>
<td>Frequency of Use</td>
<td>Regular - Casual</td>
</tr>
</tbody>
</table>

**Reference:** [Users and Workplace!]

#### Dimension: *Work Place*

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning/Training</td>
<td>Possible - Not possible</td>
</tr>
<tr>
<td>Time Pressure</td>
<td>None - Low - Medium - High</td>
</tr>
<tr>
<td>Risk</td>
<td>None - Low - Medium - High</td>
</tr>
<tr>
<td>Type</td>
<td>Office - Mobile - Factory - Public system</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal - Rugged (different types)</td>
</tr>
</tbody>
</table>
Overview of the Design Parameters

Reference: Users and Workplace!

Design Options: Tasks

<table>
<thead>
<tr>
<th>Option</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Structure</td>
<td>One-screen (Compound) - One Screen (Linear) - Multiple Screen - Wizard</td>
</tr>
<tr>
<td>Control</td>
<td>User - System</td>
</tr>
<tr>
<td>Guidance</td>
<td>None - System (Wizard)</td>
</tr>
<tr>
<td>User Support: Intensity</td>
<td>None - Basic - Intermediate - Extensive</td>
</tr>
<tr>
<td>User Support: Medium</td>
<td>None - Onscreen - Online Help - Written Documentation</td>
</tr>
<tr>
<td>Display Mode</td>
<td>Parallel - Sequential</td>
</tr>
<tr>
<td>Display Mode: Parallel</td>
<td>Multiple or additional primary windows - Multiple or additional secondary windows - Dockable windows - Parallel screen areas or tiles (Compound screens)</td>
</tr>
<tr>
<td>Display Mode: Sequential</td>
<td>Sequence of primary windows - Sequence of secondary windows - Scrollable primary window - Alternate views within a stable frame of reference</td>
</tr>
</tbody>
</table>

References: The Design Issues, User-Oriented Design Principles, Display Options for Tasks

Data (Objects)

Dimension: Formal Characteristics

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Sets: Structure</td>
<td>Unordered Set - Ordered List - Tabular Data - Hierarchy</td>
</tr>
<tr>
<td>Data Sets: Size</td>
<td>Small - Large - Fixed vs. Unknown</td>
</tr>
<tr>
<td>Singular Data Objects: Internal Structure</td>
<td>Unordered Set - Ordered List - Tabular Data - Hierarchy</td>
</tr>
</tbody>
</table>
### Overview of the Design Parameters

<table>
<thead>
<tr>
<th>Singular Data Objects: Number of Components</th>
<th>Small - Large - Fixed vs. Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singular Data Objects: Scale Type</td>
<td>Non-numeric - Numeric (different scale types)</td>
</tr>
<tr>
<td>Singular Data Objects: Abstractness</td>
<td>Abstract - Concrete (real-world)</td>
</tr>
<tr>
<td>Singular Data Objects: Value Range</td>
<td>Discrete (digital) - Continuous (analog)</td>
</tr>
</tbody>
</table>

**Reference:** Data and Their Requirements

### Structure and Navigation

**Dimension: Structure - Formal Characteristics**

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Origin of Structure</td>
<td>Flow of Processing - Data Structure</td>
</tr>
<tr>
<td>Basic Task/Object Structure</td>
<td>none - linear (sequential) - hierarchical - complex</td>
</tr>
</tbody>
</table>

**Design Options for Structure**

<table>
<thead>
<tr>
<th>Option</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Application Structure</td>
<td>Simple One-screen application - Sequence - Hierarchy - Network - Compound Screen (Complex One-screen Application)</td>
</tr>
<tr>
<td>Compound Screens: Metastructure</td>
<td>None - Index Structure - Stack- Queue</td>
</tr>
<tr>
<td>Compound Screens: Screen Structure</td>
<td>No Division - Vertical Division - Horizontal Division - T-Structure - Complex Structure; Overview-Detail Structure</td>
</tr>
</tbody>
</table>

**Reference:** Structure and Navigation - General Considerations, Structure and Navigation - How to Optimize It

**Dimension: Navigation - Formal Characteristics**

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Overview of the Design Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Macro Navigation - Micro Navigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>Within Task - Trip (to other Task, temporary) - Between Tasks</td>
</tr>
</tbody>
</table>

#### Design Options for Navigation

<table>
<thead>
<tr>
<th>Option</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movement</td>
<td>Screen Change in one Window - Screen Change to another Primary or Secondary Window - Movement to other Area on the Same Screen - Movement to View (on the same Screen)</td>
</tr>
<tr>
<td>Mechanism</td>
<td>Drag &amp; Drop - Pushbutton(s) - Menu - Toolbar - Tabstrip - (Road/Structure) Map - Path (Stack) - History List - Index - Link List</td>
</tr>
<tr>
<td>Mode</td>
<td>User-Initiated - Automatic</td>
</tr>
</tbody>
</table>

**Reference:** Structure and Navigation - General Considerations, Structure and Navigation - How to Optimize It

**Dimension: Users**

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Knowledge</td>
<td>Beginner - Intermediate - Proficient</td>
</tr>
<tr>
<td>Domain Knowledge</td>
<td>Little or none - Basic - Good</td>
</tr>
<tr>
<td>Frequency of Use</td>
<td>Regular - Casual</td>
</tr>
</tbody>
</table>

**Reference:** Users and Workplace

**Dimension: Work Place**

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning/Training</td>
<td>Possible - Not possible</td>
</tr>
<tr>
<td>Time Pressure</td>
<td>None - Low - Medium - High</td>
</tr>
</tbody>
</table>
Overview of the Design Parameters

<table>
<thead>
<tr>
<th>Risk</th>
<th>None - Low - Medium - High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Office - Mobile - Factory - Public system</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal - Rugged (different types)</td>
</tr>
</tbody>
</table>

Reference: Users and Workplace

Presentation

Dimension: Data

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Set Structure</td>
<td>None - Unordered set - List - Simple Hierarchy, Hierarchy, Network</td>
</tr>
<tr>
<td>Attribute Presentation (Internal Structure)</td>
<td>Field(s) - Single table - Parallel table - Tree</td>
</tr>
<tr>
<td>Alternative Data Presentation</td>
<td>Single - Few - Many</td>
</tr>
<tr>
<td>Format</td>
<td>Text - Graphics - Mixed</td>
</tr>
<tr>
<td>User: Domain Knowledge</td>
<td>Beginner - Intermediate - Proficient</td>
</tr>
<tr>
<td>Workplace Type</td>
<td>Office - Mobile - Factory - Public system</td>
</tr>
</tbody>
</table>

Reference: Presentation of Data

Dimension: Functionality

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Task-related - Interaction-related</td>
</tr>
<tr>
<td>Access Method</td>
<td>Mouse - Keyboard - Touchscreen (with or without pen)</td>
</tr>
</tbody>
</table>
### Overview of the Design Parameters

<table>
<thead>
<tr>
<th>Format</th>
<th>Text - Graphic - Mixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>User: Computer Knowledge</td>
<td>Beginner - Intermediate - Proficient</td>
</tr>
<tr>
<td>Workplace Type</td>
<td>Office - Mobile - Factory - Public system</td>
</tr>
</tbody>
</table>

**Reference:** Presentation of Functionality

**Source:** From Analysis to Design: Bridging the Gap
The Design Issues

Task Orientation | User Orientation | Processing and Display Requirements | Structure and Navigation | Presentation | User Support

To set our design focus we present an overview of issues that drive design. These issues are the basic “themes” for application design - all of them have to be addressed and need answers.

The first two issues, task and user orientation are "prerequisites" for this design guide. We assume that you already collected these data during brainstorming phases, reviews and site visits. As minimum requirements we expect that there exist user roles or other forms of prototypical users, like "personas", and scenarios that describe typical situations in which the software is used. Building on these data, the other issues below comprise the actual design phase, which will be covered in more detail in Part III: Defining the Design; the general approach is presented in Part I: Design Approach.

User analysis combined with task analysis provides you with the information that you need for user- and task-oriented application design. SAP has developed and adopted a set of tools and methods that developers can use for this purpose. See the respective sections in the Design Guild for more information on these topics!

One final warning: Make sure that most questions regarding users and scenarios have been settled in the analysis phase. Do not carry those questions over into the design phase. Discussions will never end, personal opinions will dominate, and the design will change like a chameleon!

Task Orientation

Applications are developed for a purpose: Users are meant to accomplish certain tasks with an application. The task determines what has to be done in an application, and therefore is the primary driving force for application design.

There are two basic issues concerning the accomplishment of tasks:

- Effectiveness: The application must enable users accomplish the task.
- Efficiency: Processing the task should be as efficient and error-free as possible.

Today’s software has the tendency to grow: Applications add features with every release, until they become a "one-for-all application", serving a multitude of purposes, situations, exceptions, customers, users etc. Users are overwhelmed by this complexity, use only a fraction of the functionality, and thus may work inefficiently or even give up.

Task orientation can help to stop this trend towards mega applications and to focus on simple, easy-to-use single-purpose applications. It implies that developers ask customers and end users about the requirements of the task, the working environment and context, the prospective user population, etc. In other words, developers should first conduct a task and user analysis before they start designing their application or even start coding it.

User Orientation

Although it should be more than evident that developers design their applications for the end users, many pieces of software might lead to a different conclusion. The gap between developers and end users often seems huge.
Here are some ideas that may help to bridge this gap:

- Know the user, visit and observe users in their daily work!
- Keep users informed about what is going on in your application, where they are, what they can do there, etc.!
- Let your application speak the users' language!
- Do not organize objects into artificial structures that the application may need for certain cases, but that users do not need (or even know)! At least, present such structures on the user interface only, when they are really needed.
- Do not define processes that are totally uncommon to users (if there is no time or possibility for training)!
- Design your application in ways that errors do not occur, or if this is not possible, so that users can easily recover from errors.

And here are some "mental" and "practical" exercises for putting on the users' perspective:

- Be your own end user and do "real" work with your application: Which assumptions did you make about the users task knowledge and about his or her system knowledge? Are the procedures efficient? Is navigation straightforward? Are the error messages understandable?
- Ask persons that you know and who are neither computer nor task experts to work with your application: Where do they have problems? What is easy for them? Do they like your application? ...

To help developers focus on end users and their needs, we collected some user-oriented design principles. See also Users and Workplaces for more information on users!

### Processing and Display Requirements

A given task imposes certain **processing and display requirements** on an application. Depending on the complexity of the task, these may be more or less demanding - for users as well as for the system. We also speak of the "distribution of the work". This distribution refers to questions like

- How can the task be split up into generalized tasks and how can these small tasks be combined to realize the whole task?
- Which task steps can be handled by the system, which have to done by the user?
- How can the resulting task steps be distributed on computer screens, which are their processing and display requirements?

See Task and Their Requirements for more information on tasks in general! In Gathering Requirements for Tasks we provide a quick & dirty method that might help you with collecting requirements.

### Structure and Navigation

The **application structure** is the **static** aspect of distributing a task on computer screens. The **dynamic** aspect of this issue is **navigation**.

The application structure emerges when the **scenario** or specification is transferred into computer-based processes. First a more abstract structure is created, which may only contain the basic modules and the paths between them. This is followed-up in a more fine-grained fashion when **processing and display requirements** are collected and put together: Processing steps to be done and objects to be displayed require screens or screen areas. Often there are many data to be displayed or steps to be walked through. As screen space is limited, information has to be split up and presented on different screens. This requires users to move between screens or screen areas during the flow of processing: That is what we call navigation (actually, macro navigation, as you will see later ...).
Structure and navigation are closely related: A complex structure usually implies a complex navigation. From the users' perspective navigation introduces unwanted complexity. Therefore, navigation should be constrained to a minimum! However, for novice and casual users a navigation that follows simple rules is preferable to a solution that requires less navigation, but is complex to use. Orientational cues and context information reduce navigational complexity and give users a sense of where there are and where they can go to.

Also navigation can be reduced psychologically by providing a stable frame of reference and exchanging only parts of a screen instead of moving to a new screen.

For more information on navigation see Structure and Navigation General Considerations and Structure and Navigation - How to Optimize It!

Presentation

Determining the basic views of the data and their interactions results in a structure and a navigational pattern for an application. After having designed these, you have to deal with the more fine-grained issues of application design, namely how you will present the data and - to some extent - the processes. Data presentation may involve decisions for presenting large and complex data sets as well as for single numbers. Note that presentation and interaction are closely related: Most presentations clearly determine how users can manipulate objects or object attributes. With respect to the processes, presentational decisions may also lead to different interaction modes.

All these decisions are determined by the data and processes on the one hand as well as by the prospective users on the other hand. Your end users may, for example, not know how to interact with certain controls or presentations so that you cannot use them in your application.

The second and even more fine-grained part of the presentation issue is how you arrange elements on the screen. For the sake of this design guide which covers the initial design phases, we do not expect this arrangement to be the final and polished one.

For more information on screen design see the SAP Style Guide or its compact version, which is included in the Design Guild!

Data presentation and screen layout are big and important issues in themselves, far too big, to be covered in this design guide. However, you find some information on presentational issues in Presentation of Data and Presentation of Functionality! You may also search the sections about visual design in the Design Guild for more information on this topic.

User Support

Especially untrained and casual users need to be supported and guided by the system. The system becomes an "external memory" for the users and provides them with the necessary information that they need to successfully complete their tasks. The conventional types of support information are often summarized under "help". Such help information can be

- general task information,
- documentation, background information,
- step-by-step instruction for procedures,
- field help (meaning of fields, input values, ...),
- tips and tricks.

Most help systems are cumbersome to use and - because of this - often not used at all. A more recent slogan is known as "software without manuals". This approach is based on a number of different ways to provide onscreen assistance for users like
The Design Issues

- descriptive labels, headers and screen or area titles,
- short instructions (step-by-step),
- maps and diagrams,
- context and navigational information,
- utilizing the affordance of interface elements.

These kinds of user support differ from help in being narrower in scope (just small pieces of information) and in usually being permanently available (or at least easily accessible). They may put higher demands on the underlying GUI technology (e.g. bitmaps, links, ...) and require valuable screen space; both reasons account for their rare use in current applications.

"Affordance" (a term coined by Donald Norman) means that interface elements tell users how they can be used. Elements differ in this respect - some are more obvious to users than others. So designers can use this feature to better match the interface to the prospective users. Sometimes screen elements actually "invite" users to use them. For example, a button tells the user "click me!", a field tells "enter a value!". Thus you can often watch a lot of clicking around when there are many buttons on a screen. Or users believe they have to enter values into all fields, not only into the relevant ones.

Two important types of such information are context information and information about the navigational options. These types of information help the user to be oriented and to keep track of their task. We cover them in more detail in User-Oriented Design Principles.

From the users' perspective support should be provided effortless, understandable and concise. Basic information needs should be served onscreen, while background information should be hidden first and may be accessible via special help systems or even written documentation (here some effort is acceptable).

For more information on users see Users and Workplaces!

Source: From Analysis to Design: Bridging the Gap
User-Oriented Design Principles

User-oriented design principles help developers to view applications from the users’ perspective and to move towards the direction of simple, user and task-oriented applications. We illustrate these principles with examples and possible interface techniques to realize the principles.

Intuitiveness, Transparency

Applications should be intuitive and transparent, especially if they address novice and casual users. Note however, that what users find intuitive or transparent, may greatly vary and depend, for example, on their prior knowledge.

Basically there are two ways to accomplish intuitive and transparent applications:

- Either the system tells users what is going on, or
- users have to know or learn it.

Making an application self-explaining is putting the informational burden on the system. Using metaphors, on the other hand, builds on pre-existing user knowledge.

Examples

- Onscreen instructions (step-by-step help, field help), onscreen diagrams showing the flow of control (overlay screen, diagram) or the application or data structure,
- user terminology,
- known organizational structures and processes,
- metaphors,
- previews showing the consequences of actions.

Techniques

- **Help area** with switchable contents; show/hide, dockable; can be realized as HTML control (step-by-step help, field help),
- **overlay screen**: a transparent layer over the screen containing instructions, captions, diagrams, or just arrows indicating the flow of control; can be “handmade” (e.g. by the user him- or herself); should be easily shown or hidden (mouse, key),
- **illustrating graphics** like pictures, figures, or diagrams on the screen.

Directness

Directness helps to reduce the psychological distance between a software system and its users: Users should accomplish their task steps as “directly” as possible. For example, to access an object's details, it is more direct to simply click a button close to the object or a link with the object's name instead of having to call a function from the menu.

Examples
Directness is often related to **simple physical interactions** with the system like

- simple clicks or point operations (instead of menu calls or command sequences),
- drag & drop operations like dragging an objects to a trash can to delete it, or dragging a slider to set a value.

Directness can be also provided on an "**informational basis**" like

- visual feedback during and after an action,
- transparent procedural steps, for instance, via previews or diagrams
- transparent navigational steps, for example, through maps or path information.

**Techniques**

- **Drag-and-drop** operations, drag operations
- **point-and-click** operations,
- **in-place functionality** (e.g. in-place editing),
- **immediate feedback** for actions,
- **preview** functionality (combined with undo functionality),
- **graphics**, image maps (context sensitive dockable areas).

**Division of Labor, Automation**

Computers should relieve users from work, not impose unnecessary work load on them. Therefore, distribute the work load between the system and the users so that users do not have to do things that the system can do for them. For example, do not require users to input data that the system could know.

In addition, many processes could be automated, or the system could make “intelligent” guesses about user actions, if it had or would use more “intelligence”. This, too, removes unnecessary work load from users.

**Examples**

- If users have to enter e.g. customer data, most of the address data could be drawn from a data base as soon as the user enters some “characteristic” data (e.g. the country and the postal code).
- The system completes user entries.
- The system anticipates user behavior like choices, often used functions or data.

**Techniques**

- **Automation techniques** like intelligent completion of entries within one field and across fields,
- **presets** for fields that need no longer be visible,
- **intelligent restructuring of control variables** to provide understandable decision options for users.

**Efficiency, Economy**

Users want to get their job done **efficiently**, fast, and without errors. This is the **number one issue** for users - you should do everything you can do in helping users in this respect! Efficiency is closely related to **economy**. Users do not want to do more steps than necessary. And they hate to be obstructed by the system.

Efficiency and economy should also be viewed from the **users perspective**, not from the system’s or the task’s perspective!
User-Oriented Design Principles

Operations that appear efficient or economical to users may not seem efficient or economical from a technical point of view. Users have their own "granularity" of sensible steps and their own opinion about mental strain. There are cases where they even use less efficient procedures, because these are more economical from a mental perspective (require less problem solving).

Also, do not require users to "help" the system. Do not implement procedures in certain ways, because this is easy to implement or easy for the system (e.g., because it minimizes data base access)! Of course, you often have to make compromises, but make them on the basis of a user-centered judgment!

Examples

- Streamline the procedures for **minimal number of steps** (sensible steps, not unintuitive macro operations!)
- do not require users to "help" the system (see "division of labor" and "automatization")
- minimize the possibilities for users to commit **errors** (errors should not occur!)

Techniques

There is not a single technique to achieve efficiency and economy. Usually a bundle of measures is needed to achieve these goals. Task and user orientation help to make sure that the important and most needed steps in a task go smooth and that the prospective users can handle them.

Reduce Complexity

The system should not "overload" users with information and functionality. It should also not hide important information and functionality from users, so that they have to search for it or may even never consider that it's there.

Examples

Especially, it should not provide

- **unnecessary information** (e.g. unnecessary fields),
- **incongruent** or **conflicting information**,  
- **confusing** or **badly structured information**, like irregular screen layouts or tables, or confusing help screens, etc.
- **unnecessary functions** (e.g. seldom or never used functions),

The system also should not introduce complexity by hiding **important information or functionality**, like

- **hiding functions** by placing often used functions into higher levels of cascading menus,
- **hiding objects** like placing object data into nested tabstrips.

In both cases, the search space can be huge, if users do not know where to look for the items.

Techniques

There is not a single technique to reduce unnecessary complexity. Usually, a bundle of measures and ongoing "streamlining" is needed to achieve this goal. Analyzing task and users and continuing user feedback help to decide which information and functionality is necessary and which is only of secondary importance or "nice to have".
Context, Orientation

Being informed about the current state is an essential human need. In the past and still today software is very poor in providing the necessary information to users. To help users to successfully accomplish their tasks, software has to act as an "external memory" and to provide the necessary cues to them. This is especially true for beginners and casual users, while proficient users may not even notice such information or find it disturbing.

There are two important types of information for users: Context information and orientational aids. Both are closely related and may be hard to separate. For this paper, we distinguish between the two in the following way:

- **Context** information refers more closely to the current state, the task, and its embedding in the general work context.
- **Orientalational aids** refer to navigation within a task or a sequence of tasks.

**Context**

Users need to be informed about the general context of their task. This information may relate to their state

- within a **task** (Which task and partial task within it am I doing?),
- within a **sequence of tasks** (work flow - through which tasks do I have to proceed in order to do my job?),
- within the **general work environment** (company, department, ... - How is my task related to other tasks in my department/company?).

**Techniques for Providing Context**

- **Screen titles and subtitles**, 
- **status information**, preferably in fixed screen areas, 
- **stable screen areas** that are dedicated to certain purposes 
- **perceptual clustering**, e.g. via colored or patterned screen areas that cluster screen elements into meaningful groups.

**Orientation**

Being oriented is an essential human need. People always strive for answers to the following questions:

- **Origin**: Where do I come from?
- **Current location**: Where am I? What can I do here?
- **Destinations**: Where can I go from here, what can I do there?

**Techniques for Providing Orientation**

- **Path information** (Where do I come from?), 
- **screen titles and subtitles** (Where am I? What can I do here) 
- **target** (destination, goal?) **information** (Where can I go from here? What can I do there?).

Currently, orientational information is presented only rudimentarily and unobtrusively so that users easily overlook it, or not find it at all:

- Typically, **target information** is provided via pushbuttons (pushbuttons may be arranged in toolbars, floating palettes, button groups or as singular buttons) and menu entries for functions. In hypertext systems, links provide target information, too.
- **Screen titles** are usually provided in the window title only and therefore out of the users’ attentional focus; subtitles are provided as group box headers.
- **Path information** is not provided in most cases.

Orientation is improved by the following methods:
User-Oriented Design Principles

- **Headers** for screens and for groups of screen elements: Headers of appropriate type (size, color, font, ...) are easily recognized by users.
- **Hypertext links** for functions and/or target screens: Links may provide better information about the target, if they are embedded into a context and if they are more wordy than button texts.
- **Road maps**, process maps, application structure maps: These maps provide overviews over a structure or sequence of steps; they also help users to find their current location and the links between different destinations.
- **Path information for back navigation** (could be provided as links or buttons): This information helps users to find their ways fast and easily back to their starting points.

### Stability, Continuity

Providing users with a stable, trustful working environment. This helps to reduce perceived complexity and thus relieves users from mental strain.

**Note:** Orientational aids and context information may also help to increase the perception of stability.

### Examples

- On a touchscreen certain screen areas are reserved for context information and for navigation functions, while the contents of one area is exchanged according to the processing needs.

### Techniques

- **Tabstrips:** Provide navigation (random access) between views while maintaining a stable environmental frame.
- **Index:** An index is a stable screen area which provides links to objects, object parts etc. for easy and fast navigation between them. An index frame resembles a tabstrip in function, but may have very different presentations. It also accommodates more flexible structures than a tabstrip: While the tabstrip just provides a linear random access structure, an index may be linear, a simple hierarchy (two or three levels) or even a complex hierarchical or network structure.
- **Screen Division:** A screen is subdivided into areas with reserved functions. Some of these areas may be exchanged according to the selected function. Depending on the implementation, the screen areas are resizable (e.g. resizable HTML frames) or of fixed size (e.g. on touchscreens or character-based terminals).

### Transparent, Efficient, and Minimal Navigation

**Navigation** is closely related to orientation and stability. The less navigation is needed, and the more stable a working environment appears to users, the better.

### Examples

- Views accessed through a screen sequence vs. random access of views via a tabstrip.
- Stack-like navigation vs. direct access through a tree which serves as an index.

### Techniques

- **Tabstrips:** Provide navigation (random access) between views while maintaining a stable environmental frame.
- **Road maps**, structure maps etc.: These maps allow to combine random access with orientational aids and context information (e.g. via screen previews).
- **Path, History List:** These devices mostly support backwards navigation.
- **Index:** An index is a stable screen area which provides links to objects, object parts etc. for easy and fast navigation between them. An index frame resembles a tabstrip in function, but may have very different presentations. It also accommodates more flexible structures than a tabstrip: While the tabstrip just provides a linear random access structure, an index may be linear, a simple hierarchy (two or three levels) or even a complex hierarchical or network structure.
Support and Guidance

The system should support users in doing their tasks on the computer. Usually it should not guide or direct them, except for complex, seldom used, or critical tasks.

Examples for Support

- Onscreen field help (for all fields on a screen or for semantically related field groups; field help),
- onscreen procedural help given as step-by-step procedures,
- onscreen information about the task and the general context of the task,
- screens that act as diagrams showing the flow of processing/control.

Techniques for Support

- Help area with exchangeable contents (e.g. for field help, step-by-step help; this area may be realized as an HTML control; it could be dockable, have show/hide buttons etc.
- overlay screen (see above),
- object-related explanations close to the object (on demand? -> see tool tips),
- tool tips, bubbles etc. that appear when the mouse is moved over objects (note: tool tips are "transient" while object-related explanation are independent of the mouse state).

Examples for Guidance

- Messages for critical errors in modal dialogs,
- modes where users cannot call other functions, e.g. a "paint" mode,
- assistants or wizards that guide users through a sequence of critical or complex steps.

Techniques for Guidance

- System prompts,
- modal dialog windows,
- modes that "lock" users in certain functions until they actively leave them,
- chains of modal dialog windows (assistants or wizards) for structuring task steps.

User-Goal Orientation

Users have goals, one of them - but not the only one - is to accomplish their tasks and to do this efficiently. But users may also have further goals that even may conflict with the task goals or company goals (see Cooper).

For developers it is important to know these goals and to design applications accordingly. One important user goal is not to look stupid. Therefore, help users in accomplishing their task, do not make your application a riddle or puzzle for them!

Task orientation is closely related to efficiency, because it does not suffice that a software somehow makes it possible for users to accomplish a task. Users want to feel comfortable with the system and to do their task with a reasonable amount of effort, not with an insurmountable one.

Examples

- When users enter a screen, they want to see immediately whether this screen serves their task goals, and how they can accomplish their task there.
- Unnecessary steps are a symptom for missing task orientation. They reduce the users' efficiency and annoy users.
User-Oriented Design Principles

- Unnecessary information are a symptom for missing task orientation. They distract users and thus reduce their efficiency.

Techniques

- **Goal "Keep me informed"**: Provide context information (see there) to keep users informed about the task, its state and the steps to accomplish the task (e.g. a web application might use group headers as instructions).
- **Goal "Do not let me look stupid"**: Provide affordances and onscreen help (see there) to make your application self-explaining; prevent or reduce the possibilities of errors and error messages.
- **Goal "Let me do it efficiently"**: Design efficient procedures, know the task and the users, design for the relevant cases, handle less relevant cases and exceptions differently (hide them, if possible).

Error Robustness

**Error** is something that **should not occur**! Users do not want to make errors, and they do not want to look stupid - not to other people and not to machines. So, making no errors is one of the primary user goals. Help users to accomplish this goal!

Examples

- Having entered a postal code for a city, the user has to enter the city; the user misspells the city and gets an error message. In this case, the city is uniquely defined through the postal code. The system should fill it in automatically, and thus prevent that users can make an error here.
- Having entered the city, the user has to enter the postal code. The user enters a wrong postal code and receives an error message. In this case, the postal code is either uniquely defined or limited to a small number of postal codes. The system therefore should present these as a selection to the user, thus again preventing that the user can commit an error.
- If you have to provide error messages, check that these message "speak the users' language" and are easily understood.
- If possible, help users to recover from errors!

Techniques

- **Error prevention**: Limit choices to the ones that make sense. This refers to function selection as well as to data entry. If choices depend on other choices or options, disable unavailable choices accordingly.
  - **Limit function choices**: Deactivate functions that make no sense in a given situation by graying them out in the menus and toolbars or by hiding them.
  - **Limit number of values for data entry**: Use drop-down lists or dialogs where users can select correct values (of course there is much work to be done on your side to preselect these values).
- **Error prevention**: Prevent entry of **redundant** values or values that the system **already knows** or could know.
- **Error recovery**: Help users to recover from errors by providing the opportunity to re-enter values etc. This may require holding data before submitting them.

Source: From Analysis to Design: Bridging the Gap
Tasks and Their Requirements

Breaking Down the Task into Elementary Tasks | Recombining Elementary Tasks: Grouping and Distribution

This section and the following ones, Gathering Requirement for Tasks and Display Options for Tasks are devoted to the task-related aspects of application design. Here we present the more general aspects, while the other two sections are more "practical".

Overview of the Criteria and Possible Values for Tasks

Because the overview is rather long, we transferred it to an extra page!

Breaking Down the Task into Task Patterns and Elementary Tasks

In the end all user interface designs have to come up with a selection of interface elements, that are combined and arranged on screens, so that users can accomplish their task. A bottom-up procedure would just start with putting controls on the screen, hoping that somehow these elements "magically" combine to a coherent whole that is a joy for users to work with. A top-down procedure, on the contrary, is a bit "top-heavy" and delays the implementation work as far as possible. In Elements of Application Design we outline such a top-down procedure: It breaks down a task into steps and looks for a mapping between task steps and interface elements. In the following we describe this procedure in more detail.

Task Patterns and User Interface Patterns

In this guide we often use the term "pattern". Patterns are reusable and recurring - prototypical - elements; they can compose a hierarchy: Elementary patterns can be assembled into compound patterns or high-level patterns be broken down into smaller components. There is a hierarchy of patterns on the task side, where tasks can be broken down into task steps or task patterns (if the steps are prototypical) and further into elementary tasks. Or, the other way round, elementary tasks can be combined to task patterns, subtasks and tasks. The same is true on the interface side, where we find user interface patterns of varying complexity, comprising simple to complex "prototypical" combinations of screen elements.

Mapping Real-World Tasks Onto Interface Patterns

To find interface solutions for task steps, we propose to establish a correspondence between task steps or patterns and interface patterns; this can be done by mapping task steps or patterns onto user interaction patterns first. This is an abstraction of the steps, that is done by "stripping off" the task-specific aspects of the task steps and reducing them to interaction patterns which are the building blocks of all interactions. Such building blocks must be independent of the task they describe or model - they must be "generic". We propose a preliminary set of generic interactions patterns for mapping specific tasks onto:

- Enter Data
- Inspect Data (+ Compare Data)
- Manipulate Data (Move, Insert, Delete, Copy, Change, ...)
- Search
- Browse
- Select ...
  - Data
  - Functions
  - Options
- Initiate Action(s)
Tasks and Their Requirements

- Decide
- Get Help

For a description of these generic interactions see "Interaction Patterns" in Part IV: Background Information! The example below, looking up a colleague’s address, shows how this mapping can be done in a table:

<table>
<thead>
<tr>
<th>Screen or Screen Area</th>
<th>Task-Specific Step</th>
<th>User Interaction Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log-on screen</td>
<td>Enter Name and password</td>
<td>Enter data</td>
</tr>
<tr>
<td></td>
<td>Press &quot;Log-on&quot; button</td>
<td>Initiate action</td>
</tr>
<tr>
<td>--- Screen change ---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search area of screen</td>
<td>Enter Name of colleague using certain fields (e.g. last</td>
<td>Enter data</td>
</tr>
<tr>
<td></td>
<td>name, first name, etc.)</td>
<td></td>
</tr>
<tr>
<td>Hit list area below search area</td>
<td>Scroll hit list and search for colleague</td>
<td>Browse data</td>
</tr>
<tr>
<td></td>
<td>Select line where colleague is displayed</td>
<td>Select data</td>
</tr>
<tr>
<td></td>
<td>Press &quot;Details Button&quot;</td>
<td>Initiate action</td>
</tr>
<tr>
<td></td>
<td>Quit application</td>
<td>Initiate action</td>
</tr>
</tbody>
</table>

Table: Mapping specific actions onto interaction patterns for the task “Find Address”

Another approach to mapping is to model the task as a flowchart. This procedure also helps to abstract from the concrete task and to map the task steps onto generic actions, namely interaction patterns.

So, What Is the Benefit of This Mapping?

The purpose of the mapping is to simplify the process of finding correspondences between task steps and interface patterns. Stripping off the task-specific components of actions and thus moving towards interaction patterns makes it easier for us to find prototypical interface patterns for certain task steps. If we learn, for example, that entering a name can be “reduced” to an “enter data” interaction, we realize that the data entry aspect is prominent in this task step, and we also know that there are interface solutions for it - usually not one, but several among which we can choose the most appropriate for the given context.

Mappings: Many or Just One?

There is rarely a direct one-to-one correspondence between task patterns and interface patterns. The choice of interface elements for, say, a certain elementary task like browsing for people in a hit list, depends heavily on the context of the task, the data involved, the user types and the physical environment. There are too many restrictions and boundary conditions to be observed that interface design can be reduced to a simple “table-lookup”. For a given task there usually exist several alternative
user interface patterns, the pros and cons of which have to be evaluated on the basis of the most important task features as described in the scenarios. This is where the design criteria and their values come in. They help to find unique solutions by excluding certain designs that do not fulfill the requirements.

**Ready-Made Solutions**

There are prominent cases, where a direct correspondence does exist and where it is fairly easy to pick a proven solution. These "ready-made" solutions help to standardize user interfaces and simplify the interface designers' work as well as the users', who can work with familiar designs and reuse their knowledge. Often these proven user interface patterns can be reused in a variety of tasks.

Examples for such "prototypical" user interface patterns are frequently used dialogs like the print dialog, selection screens, the shuffler (from Cooper) for filtering data, or the "explorer" tree as access mechanism.

**Recombining Elementary Tasks: Grouping and Distribution**

After a task has been broken down into one or more sequences of elementary steps, the question is how these steps can be grouped into meaningful subtasks, and how these subtasks can be distributed on screens or screen areas in order to achieve an efficient flow of processing and to accommodate the prospective users.

Let us illustrate this with a simple example! A basic master data application involves the following steps and side-paths:

- Specification of new or existing object (may involve browsing or search),
- specification of views,
- entering, editing or displaying the data,
- processing of detail data,
- saving the data, return to the starting point.

Typical design questions might be:

- Should specification and editing be displayed simultaneously?
- Should items be processed in parallel or sequentially?
- Should the normal view and the detail view be displayed in parallel?

In Gathering Requirements for Tasks we put this procedure to life by proposing a method for collecting the processing and display requirements for a given task. We complement this methods by proposals for using presentations and defining priorities for subtasks. In Display Options for Tasks - An Overview we also present an overview of possible design options under the assumption that data have to be processed either sequentially or in parallel. Note also that helpful additions can be found in Part IV: Background Information.

**Source:** From Analysis to Design: Bridging the Gap
## Overview of the Criteria and Possible Values for Tasks

### Dimension: General Task Characteristics

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus</td>
<td>Editing - Inspecting - Browsing</td>
</tr>
<tr>
<td>Objects</td>
<td>One (Simple or Complex) - Many Simple - Many Complex</td>
</tr>
<tr>
<td>User Support</td>
<td>None - Basic - Extensive</td>
</tr>
</tbody>
</table>

### Dimension: Space (Processing and Display Requirements)

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing and Display Needs: One vs. Several Items</td>
<td>Access to Multiple Objects/Object Parts Necessary - Access to Single Access to Objects/Object Parts Possible</td>
</tr>
<tr>
<td>Processing and Display Needs: Overview - Detail</td>
<td>Needed in Parallel - Not Needed in Parallel - Not Applicable</td>
</tr>
<tr>
<td>Data Objects</td>
<td>Single Attributes - Multiple Attributes for One Object - Single Attribute for Many Objects - Multiple Attributes for Many Objects</td>
</tr>
<tr>
<td>Presentation</td>
<td>Needs Little Space - Needs Much Space</td>
</tr>
<tr>
<td>Support Information</td>
<td>None - Yes, Needs Space</td>
</tr>
</tbody>
</table>
### Dimension: Time (Processing and Display Requirements)

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel Access to Objects or Object Parts: Duration</td>
<td>Never - Temporary - Permanent</td>
</tr>
<tr>
<td>Processing Time</td>
<td>Irrelevant - High Speed/Volume Processing</td>
</tr>
</tbody>
</table>

### Dimension: Users

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Knowledge</td>
<td>Beginner - Intermediate - Proficient</td>
</tr>
<tr>
<td>Domain Knowledge</td>
<td>Little or none - Basic - Good</td>
</tr>
<tr>
<td>Frequency of Use</td>
<td>Regular - Casual</td>
</tr>
</tbody>
</table>

For the dimension work place and for details on users and work places see Users and Workplace!

### Design Options: Tasks

<table>
<thead>
<tr>
<th>Option</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Structure</td>
<td>One-screen (Compound) - One Screen (Linear) - Multiple Screen - Wizard</td>
</tr>
<tr>
<td>Control</td>
<td>User - System</td>
</tr>
<tr>
<td>Guidance</td>
<td>None - System (Wizard)</td>
</tr>
<tr>
<td>User Support: Intensity</td>
<td>None - Basic - Intermediate - Extensive</td>
</tr>
</tbody>
</table>
Design Methodology

<table>
<thead>
<tr>
<th>User Support: Medium</th>
<th>None - Onscreen - Online Help - Written Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Mode</td>
<td>Parallel - Sequential</td>
</tr>
<tr>
<td>Display Mode: Parallel</td>
<td>Multiple or additional primary windows - Multiple or additional secondary windows - Dockable windows - Parallel screen areas or tiles (Compound screens)</td>
</tr>
<tr>
<td>Display Mode: Sequential</td>
<td>Sequence of primary windows - Sequence of secondary windows - Scrollable primary window - Alternate views within a stable frame of reference</td>
</tr>
</tbody>
</table>

User support, guidance and control are discussed in The Design Issues and User-Oriented Design Principles. For details on option display mode Display Options for Tasks!

Back to Tasks and Their Requirements, back to Gathering Requirements for Tasks

Source: From Analysis to Design: Bridging the Gap
Gathering Requirements for Tasks

Collecting the Basic Processing and Display Requirements | More Complex Tasks | Extension: Paper Clippings | Extension: Prototypes | Focus of User Actions | 80/20 Analysis

We suggest a method for finding task requirements and possible design solutions that is outlined below. It breaks down the task into subtasks and simple task steps (elementary tasks or interaction patterns) and lists the requirements imposed by the task steps and the processed data. It also lists possible presentations and may optionally include priorities.

This procedure can be supported by external aids like paper clippings, sketches, or graphic tools - even crude paper prototypes can be built. For more complex applications, start with "submodels" and integrate them afterwards.

We complement this method with two strategies for focusing the design: (1) Asking for the focus of the user actions helps to find compromises between conflicting requirements. (2) Asking for priorities helps to rank the importance of subtasks and their related functionality and features with respect to the users and context of the task.

Collecting the Basic Processing and Display Requirements

Requirement analysis lists the subtasks that comprise the real-world task in a table in order to find out

- which subtasks and primitive steps therein comprise the task,
- which data are processes by the subtask
- which tasks should be handled and thus displayed in parallel and which can be done one after the other,
- which information should be displayed for a longer period of time and which can be displayed only temporarily,
- possible presentations,
- optional: priorities for subtasks or steps.

Steps

The primitive steps can be elementary tasks or if you want to drive the decomposition even further, interaction patterns. It is not that important which level of abstraction you use for the steps, but you should be consistent within your tables. Below you find an example for such a table!

Design Questions and Solutions

For design decisions now the dimensions, criteria and values for tasks, data and possibly presentation come into play: You may scan the overviews of the criteria and their values for relevant design criteria and add them to a check list. The properties and dimensions characterize the "domain", while the criteria highlight the problems and "ask the questions"; values are possibilities for criteria. The sections about data, presentation, users and work place provide more information and, in some cases, possible answers. This guide is up to now not able to offer solutions to most design questions, and probably never will. But more important is in first place to state possible problems; knowing the problems can help teams to discuss and find solutions on their own.

Extending the Table

You can also use these criteria to extend the table with relevant criteria. For example, the table that we propose already contains columns for some temporal and spatial requirements.
Gathering Requirements for Tasks

More Information...

For more information on elementary tasks and interaction patterns see Elementary Tasks and Interaction Patterns. For possible presentations, their uses, advantages and disadvantages see Display Options for Tasks - An Overview!

Overview of the Design Parameters provides you with an overview of all design parameters in this guide. Click here to find the parameters for tasks. Note that some of the criteria are relevant, when the more fine-grained design starts and not in this initial stage. More information on data, presentation, users and work place can be found all over Part III: Defining the Design.

Requirement analysis can be carried out with the development team in a design session. You find more information on design sessions in the respective section in the Design Guild!

Example: Creating, Editing or Displaying a Debitor

<table>
<thead>
<tr>
<th>Subtask</th>
<th>Processing (Steps)</th>
<th>Data</th>
<th>Display Subtask in Parallel to...</th>
<th>Display Subtask Temporarily?</th>
<th>Possible Presentation</th>
<th>Optional: Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification</td>
<td>● Enter specification data (few)</td>
<td>● Object name</td>
<td>Browsing, Search, Hit List (if available)</td>
<td>Could be displayed temporarily</td>
<td>Main window, dialog window, header area</td>
<td>always necessary</td>
</tr>
<tr>
<td></td>
<td>● Get reference object (from list?)</td>
<td>● Object set for selecting object or reference object</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Browsing</td>
<td>● Browse list of existing data and select object</td>
<td>● Object set for selecting object or reference object</td>
<td>Specification, Search, Hit List (if available)</td>
<td>Unclear</td>
<td>List in main window</td>
<td>always necessary</td>
</tr>
<tr>
<td></td>
<td>● Filter list</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search</td>
<td>● Search object(s) according to incomplete specification = enter search criteria</td>
<td>● Object name (incomplete)</td>
<td>Specification, Browsing, Hit List (if available)</td>
<td>Unclear</td>
<td>Fields in main window (area)</td>
<td>always necessary</td>
</tr>
<tr>
<td></td>
<td>● Display hit list (search results) after search</td>
<td>● Additional attributes (for search)</td>
<td></td>
<td></td>
<td>List in main window</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Hit list</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Selection in Hit List</strong></td>
<td><strong>Itemize Processing</strong></td>
<td><strong>Itemize Detail View</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------</td>
<td>------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>● Browse hit list and select object</td>
<td>● Enter important data for item</td>
<td>● Enter detail data here</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>● Hit list</td>
<td>● Hit list Specification, Browsing, Search</td>
<td>● Return to main view</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specification, Browsing, Search</td>
<td></td>
<td>● Object attributes (most important)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unclear</td>
<td>Detail view</td>
<td>Main processing view</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>List in main window</td>
<td>No</td>
<td>Could be displayed temporarily in parallel to or instead of main processing view</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>always necessary</td>
<td></td>
<td>Fields in main window (area)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fields in main window (area)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This table should be filled in in a **coarse** manner only. It can be used during brainstorming sessions or in discussions with colleagues. In the beginning this table should list **first ideas** and **alternatives**. As the discussions proceed, the table should "**stabilize**" and present **decisions** as well as **open questions**. Mark both as such! After the discussion phase, the table can also be used to **present the final design decisions**.

The optional column **priority** originates from the **80/20 analysis** (see below) which can be carried out as part of the requirement analysis or separately. This is better suited to more complex applications, while for simple applications it may suffice to decide on the **focus** of the users' activities.

### More Complex Tasks

The example above consists of simple tasks only, where usually just one object or object sets is involved. In tasks where objects are put in relation to each other, e.g. are added to structures, moved between structures etc. the display requirements are more demanding. Such tasks may require that two or more objects or object sets are available in parallel, especially if a more "direct" interaction is intended.

**Example**: Drag-and-drop of objects between two data set requires that both set are visible at the same time. Cut-and-paste, on the contrary, only requires one data set to be visible at a time (though a parallel display would be more efficient, provided there is enough space ...).

However, the basic strategy should still work, but might require you to create several tables which finally have to be integrated, maybe into a high-level table.
Gathering Requirements for Tasks

Extension: Paper Clippings

An extension to the method described above is to write the subtasks on paper clips and to arrange them on a board. Start with placing the clippings one below the other, then group them according to the screens and screen areas they belong to. This, too, may help in deciding which information is needed in parallel and which can be displayed sequentially or only temporarily. You could also use a clip for each table cell (use different colors for the columns!).

For more complex tasks you should create several clusters of clips, one for each subtask. Then link these clusters to indicate their integration.

Instead of paper and pencil or paper clippings you can also use graphic tools for this task. The drawback is that this usually takes more time, because people put too much effort in tidying up the diagrams. On the positive side, a computer diagram is a good means for communicating the design to other people.

Extension: Prototypes

The paper clippings method - or its computer-based versions - are already close to creating a crude paper prototype. Many developers are reluctant to write down tables like the one in the requirement analysis. Instead, they prefer to put their hands on the computer and start coding. Paper prototypes might be a compromise that keeps developers from coding too early. Note, however, that the table contains information that is not contained in the prototypes. We would rather see these very rough first prototypes as a complement to the table that helps the team to better imagine the new design proposal.

Focus of User Actions

Often tasks or subtasks have different or even conflicting requirements which are hard to integrate into one coherent design. So the optimal compromise between the requirements has to be found. One strategy to find such a compromise is to ask for the focus of the users’ activities. Here are some focus questions, you may find additional ones:

- What is the basic goal of the users?
- What are the users supposed to achieve (purpose)?
- Do users spend most of the time for browsing data or for editing data?
- Do users enter few data or many data when they process an object?
- Need users to access objects in random order or does a sequential access suffice?
- Need users the objects presented in parallel, e.g. simultaneously, or does a sequential presentation suffice?
- How much screen space is needed for processing the object(s)? Is there room enough for context and orientational information?
- Do users need supporting information on the screen (which also reduces the processing space)?
- If many objects are being processed: How is the structure and the size of the data set and which representation is the most appropriate?

80/20 Analysis

For more complex tasks the 80/20 analysis can complement the requirement analysis or can be part of it. It requires to decide which activities are important for users and which are less important or just exceptions. Another possibility is to ask, which activities are necessary and which are optional or "nice to have". This should be done by ranking the subtasks with respect to the users’ global goals and to the accomplishment of the application’s purpose. Possible rankings might be "always done", "rarely
Gathering Requirements for Tasks

done", "exception", etc., and for the second ranking "necessary", "advanced feature", "optional feature", etc.

This procedure helps to focus the application design on the important activities, to make "the important things" easy, and the less important things less easy for users. For example, using the results of this analysis, you can provide easy access to often used functions and hide advanced or rarely used functions. This speeds up often used tasks and reduces interface complexity for the the standard cases.

**Note:** Optimizing does not mean that everything gets better! Usually it speeds up only the overall performance.

---

**Source:** From Analysis to Design: Bridging the Gap
## Display Options for Tasks - An Overview

**Display Mode: Sequential | Display Mode: Parallel**

Below we present an overview over display options for "real-world" tasks. Based on the distinction between **sequential** and **parallel** display, we list possible design options, their advantages as well as their disadvantages, and give recommendations for the respective options.

### Display Mode: Sequential

The "pieces" or units of information are displayed in a sequential fashion, that is, one after the other. The different techniques vary - among others - with respect to how the information is accessed.

<table>
<thead>
<tr>
<th>Design Option</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence of primary windows</td>
<td>Simple to understand, only one main window needed.</td>
<td>Usually no orientation for users, task is segmented.</td>
<td>Use for simple applications, web applications, especially for beginners. Provide users with context and navigational information!!!</td>
</tr>
<tr>
<td><strong>Example</strong>: Standard R/3 screen sequence</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Sequence of secondary windows | Keeps some of the context if primary window is still visible. | Usually no orientation for users, task is segmented. | Use only for tasks where system guidance is necessary or useful. Provide enough context information! |
| **Example**: Assistants, wizards | May be used to guide users through a task (modal windows), because the system is in control. | Loss of control may be a problem for users in certain situations. |                                                     |
### Scrollable primary windows

**Example:** Long dynpros, web pages

- Bigger tasks are on one screen and are not segmented; may therefore lead to efficient processing.
- Screens may be crammed with information.
- Users usually do not want to scroll.
- Important information may be overlooked, if it is in the hidden part of the screen.

This technique can be used if the screen has the character of a paper form. Otherwise use this technique for power users only where efficiency is the primary goal.

### Alternate views within a stable frame of reference

**Examples:** Tabstrip, index layout

- The stable frame of reference provides a context and the impression of stability to users.
- Navigation is simple and fast.
- For index layout, various structures for the object set are possible.

Tabstrip: Makes sense for a limited number of views only. The views cannot be structured.

Nested tabstrips may hide important information for users!

Index: May not fit the screen and therefore have to be scrolled vertically and/or horizontally.

Index layouts provides an efficient presentation for the following tasks:

- fast browsing through objects, object parts or information structures
- a stable frame of reference

Use tabstrips if users need:

- random access to (few) views,
- a stable frame of reference

Do not use tabstrips if users have to proceed through a fixed sequence of screens.

### Display Mode: Parallel

Here two or more information units are displayed in parallel. However, often it is only possible to display some part of the information units in parallel, not the whole set.

<table>
<thead>
<tr>
<th>Design Option</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Multiple or additional primary windows</strong></td>
<td>Context is kept.</td>
<td>Complex and possibly intransparent.</td>
<td>This layout is more suited to professional users who &quot;know what they are doing&quot;.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>Example</strong>: Multiple web pages in a web browser</td>
<td>More information can be displayed at the same time.</td>
<td>Information may be obscured.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Users can proceed in a flexible manner.</td>
<td>Window handling may take up much time and effort.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Focus of attention as well as processing path may be unclear.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Multiple or additional secondary windows</strong></th>
<th>Context is kept.</th>
<th>Visually more complex than single screens.</th>
<th>Modal dialog windows:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong>: Secondary windows for subtasks in R/3 (modal); amodal search dialog on word processor</td>
<td>More information can be displayed at the same time.</td>
<td>Information may be obscured.</td>
<td>- Use them for temporary tasks where the system needs to guide users.</td>
</tr>
<tr>
<td></td>
<td>The primary screen is kept &quot;clean&quot; from side information.</td>
<td>Windows handling may take up some time and effort, especially with amodal secondary windows.</td>
<td>Amodal windows:</td>
</tr>
<tr>
<td></td>
<td>Side paths are possible in a flexible manner, if secondary windows are amodal.</td>
<td></td>
<td>- Use them for side paths or tasks in order not to clutter the primary screen. More suited for professional users.</td>
</tr>
<tr>
<td></td>
<td>The system may guide users on side paths, if modal secondary windows are used.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dockable windows</td>
<td>Context is kept.</td>
<td>Visually more complex than single screens.</td>
<td>Useful if there is one larger primary window with several &quot;satellite&quot; windows for secondary tasks (e.g. attribute windows).</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------</td>
<td>------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Example: Visual Basic development environment</td>
<td>More information can be displayed at the same time.</td>
<td>The windows may be too small for effective work.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The primary screen is kept &quot;clean&quot; from side information.</td>
<td>Focus of attention as well as processing path may be unclear.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Less effort for window handling than with independent windows.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parallel screen areas or tiles (Compound screens)</th>
<th>Context is kept.</th>
<th>Visually more complex than single screens.</th>
<th>Tests show that proficient users work more efficiently with this layout; beginners, however, have problems with respect to complexity and flow of control.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples: Web pages with frame layout, Cooper Design proposals</td>
<td>More information can be displayed at the same time.</td>
<td>The screen areas or tiles may be too small for effective work.</td>
<td>Focus &amp; context methods should decrease the problems of small tiles as well as of missing focus of attention and processing path.</td>
</tr>
<tr>
<td></td>
<td>The primary screen is kept &quot;clean&quot; from side information.</td>
<td>Focus off attention as well as processing path may be unclear.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Less effort for window handling than with independent windows.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data and Their Requirements

Structure and Size of Data Sets | Internal Structure and Size of Objects | Combinations | Requirements: Time and Space, Presentation | Abstract vs. Concrete | Discrete vs. Continuous

Data are complementary to tasks, because any task processes at least some data. Like tasks, data impose certain processing and display requirements on an application. In the following we discuss which dimensions and characteristics of data are relevant to application design.

Presentational issues are not within the scope of this design guide, but you find a short treatment in Presentation of Data.

Overview of the Criteria and Possible Values

Dimension: Formal Characteristics

For this discussion be distinguish between singular data objects and data sets. Singular data objects are characterized through a number of attributes which can be simple values or data objects themselves. Simple values may be numeric or non-numeric (names), which refers to their scale type. Objects may be also be members of data sets.

Then there are more content-related criteria like abstractness and value range, that also help to choose an appropriate presentation that users easily understand.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Sets: Structure</td>
<td>Unordered Set - Ordered List - Tabular Data - Hierarchy</td>
</tr>
<tr>
<td>Data Sets: Size</td>
<td>Small - Large - Fixed vs. Unknown</td>
</tr>
<tr>
<td>Singular Data Objects: Internal Structure</td>
<td>Unordered Set - Ordered List - Tabular Data - Hierarchy</td>
</tr>
<tr>
<td>Singular Data Objects: Number of Components</td>
<td>Small - Large - Fixed vs. Unknown</td>
</tr>
<tr>
<td>Singular Data Objects: Scale Type</td>
<td>Non-numeric - Numeric (different scale types)</td>
</tr>
<tr>
<td>Singular Data Objects: Abstractness</td>
<td>Abstract - Concrete (real-world)</td>
</tr>
<tr>
<td>Singular Data Objects: Value Range</td>
<td>Discrete (digital) - Continuous (analog)</td>
</tr>
</tbody>
</table>

As time and space requirements for data depend very much on the task, on the data presentation that is selected according to the data's formal characteristics, the users' knowledge and maybe work place restrictions, we restrict the discussion to the formal aspects of the data and take only a short glimpse at the requirements. This keeps the discussion simpler and avoids redundancies.
Task requirements are covered in Tasks and Their Requirements, presentational requirements are covered in Presentation of Data. Requirements originating from users and workplaces are discussed in Users and Work Place.

Structure and Size of Data Sets

Objects may be members of a data set which may have a more or less complex structure, for example:

- simple unordered set: list
- ordered list: list, table
- tabular data (two and more dimensions): tables, set of tables, hierarchical tables, trees
- hierarchy: trees, hierarchical lists or tables

Each of these structures has a "typical" presentation which is shown above for each type. But - depending on the contents of the data - there may be alternative presentations available, see Presentations for more details.

Another important parameter is the size of data sets: For small sizes quite different design solutions can be used than for large or unknown numbers.

Depending on the task, it may be necessary or appropriate to process or display

- the whole data set,
- just parts of it,
- or only single elements.

What is displayed not only influences how much screen space is needed for the data display, but also the form of the presentation. For instance, single objects are usually displayed in a template-style layout, while for multiple data tables are the typical solution. Moreover displaying the whole data set may require filters to reduce the amount of data shown.

In addition, it may also be necessary to compare or process different data sets or different parts of a data set in parallel which not only requires more screen space but also way to handle these.

Internal Structure and Size of Objects

Objects may only be referred to by their names, but usually also their internal structure is relevant for the processing. For instance, a requisition consists of a list of items; an item again may consists of a number of attributes. Other objects may even have a more complex internal structure: for example, a part list may be hierarchical.

Similar to object sets, the internal structure of an object determines how its attributes are typically displayed:

- simple unordered set: list
- ordered list: list, table
- tabular data (two and more dimensions): tables, set of tables, hierarchical tables, trees
- hierarchy: trees, hierarchical lists or tables

See Presentations for more details! Also note that for presentation purposes it may be irrelevant, whether a set or the internal structure of an object is displayed.
Again, the **size** of the internal structure, that is, the **number of the components or attributes** is important for design. Like with data sets, small numbers allow for very different design solutions than large or unknown numbers.

**Combinations**

The aspects discussed so far often appear in **combinations**: Objects may have an internal structure, and on the other hand can belong to a data set which itself is structured.

**Example**: Objects may consist of a set of attributes. On the other hand, these objects may belong to an unordered set which may be sorted according to different criteria. Both of these structures have to be presented for the processing: The object set could be presented as an overview list which is used for browsing the data. The objects themselves might be presented as single screens with input/output fields, that is, as detail views where the editing is done. One open issue might be, whether both views should be presented in parallel, alternatively, or whether one set should be presented permanently and the other temporarily (and which one...).

As you see, here the necessity of different **views** for different parts or aspects of the data comes into play. These views lead us to the next issue, namely presentation requirements!

**Requirements: Time and Space, Presentation**

For finding **requirements** we take a more abstract view and ask:

1. **Parallel Display (Overview - Detail)**: Does the task require the parallel presence of all aspects of the data, e.g. of overview (data set) and details (attributes)?
2. **Parallel Display (One vs. Several Items)**: Does the task require the parallel presence of several data sets and/or objects?
3. **Duration**: Can some of the parallel displays be presented only temporarily or is a permanent display required? Which is the one to be displayed temporarily?
4. **Presentation**: Which presentation is the most useful for the data set, the processing method, and the prospective users?
5. **Screen Space**: How much space does the optimal presentation need? If space is scarce, which alternatives use less space?

Note however, that these issues are by no means independent. Decisions with respect to the presentation may in turn influence the other issues and lead to a totally different design.

We discuss time and space requirements in more detail in ????; presentational issues are found in **Presentation of Data**.

**Scale Type: Non-numeric vs. Numeric values**

Now we touch on a more complicated issue, namely the **scale type** of data. This dimension may sound academic, but it implies which math (e.g. statistics) can be done with numbers and which are the correct presentations for them. With the more widespread use of graphics it becomes harder to choose data presentations that do not fool **users** so that they come to wrong conclusions.

**Non-numeric** values are like **labels** (or names). Numbers can also be assigned to non-numeric values, but in this case numbers are like labels - you cannot do arithmetic with them (sometimes such arbitrary numbers are used for ordering items, which as well may not make sense).
Example: The numbers on the back of soccer players are labels (names).

Numeric values can have different scales (ranks vs. absolute and relative scales). Depending on the scale type, different ways of doing arithmetic are allowed.

Examples: Top ten numbers are ranks; prices, quantities or weights are "real numbers".

Example: It does not make sense to make a pie chart or bar chart from ranks like the top ten products; it does however make sense to create a pie chart or bar chart from the percentage of market share or a bar chart for the absolute volumes (which can be added to the pie chart as labels).

Tip: There are lots of textbooks covering the correct use of diagrams for data visualization!

Abstract vs. Concrete

Abstract data can often be presented in abstract notations only, and thus your design options may be restricted to text-based or other generic presentations like fields, lists, tables or trees. Data that correspond to concrete objects (real-world objects) have more presentational options: They can be displayed using abstract presentations, but in many cases other, for example image-oriented, presentations are more appropriate.

However, there is often a strong interaction between content-related properties like the concreteness and formal properties like the number of attributes: The number of data or attributes may be too large for presenting the data graphically. In other cases abstract data may be converted into more concrete ones, for example by counting the elements and displaying absolute or relative amounts.

Example

- Abstract notation: The states of the USA can be presented as a two-level list: level one displays the state names, level 2 displays attributes like, population, capital etc.
- Graphical notation: The states of the USA can be presented as a map as well: A map can display analogue features like the states' locations and sizes, as well as abstract textual information like names and populations of the states.

With respect to users you should keep in mind that while more educated users may understand abstract notations and presentations, less educated users only understand simple and image-like ones.

Discrete vs. Continuous

While on a computer all data are digital, there is a fundamental difference between quantities that are discrete and such that are continuous. This difference comes into play when other presentations as texts or numbers are used.

Examples for discrete data: Non-numeric values (names, labels), discrete value sets (like 1 to 10)

Examples for continuous data: Areas, volumes, weights, frequencies, percentages

When graphical presentations are used, discrete objects are typically displayed as discrete objects or object aggregates.
Example: Numbers can be presented as boxes or images that are clustered into groups of 5 or 10.

Continuous quantities are typically expressed by physical analogues like lengths, areas, arcs, directions etc. Note that people differ in their ability to correctly judge physical dimensions; there are also differences between the dimensions themselves.

Example: Sales volumes can be displayed as blocks (area), circles (area) or bars (length) of according size.

Larger number ranges (e.g. 1-100) can be displayed like continuous values, even if the values are discrete, because visually there is no difference to continuous values (the legend however should indicate this).

In some cases, for instance for qualitative judgments, it may on the contrary be more useful to pick some "prototypical or representative values" that cover the value range of interest, e.g. "small", "medium", "large" or "bad", "fair", "good". In other cases not the whole data range is of interest, but certain critical values or thresholds that have to be distinguished from the rest of the values or continuum.

Source: From Analysis to Design: Bridging the Gap
Structure and Navigation - General Considerations


In the following we discuss structural and dynamical (navigational) aspects of application design like where do structures come from, which are typical structures, and what does navigation mean. You find practical hints on how to optimize structure and navigation of your application in Structure and Navigation - How to Optimize It.

Overview of the Criteria and Possible Values

Because the overview is rather long, we transferred it to an extra page!

Where Do Application Structures Come From?

Application structures - and the corresponding navigation - are based on:

- the processing requirements,
- the structure of the underlying data.

Flow of Processing

The flow of processing usually determines the application structure, if

- one object is processed at a time,
- this object or its processing is relatively complex.

For example, an object may consist of components, that are processed step-by-step in a certain order.

In some cases object structure and flow of processing are identical, in other cases they are different.

If the flow of processing determines your application structure, the structure is only implicitly defined and usually hidden. The transitions from screen to screen or from web page to web page are provided through function calls via menus or pushbuttons.

Data Structure

This case is more typical for applications which process a set of objects. Here, the structure of the set of data determines the application structure. The processing of single objects often is relatively simple, while navigation between objects (browsing) plays an important role in the application.

If the data structure determines the application structure, an index structure is more appropriate. By displaying the index or the whole set of navigational options, the application structure is revealed, and thus made more transparent. The index also serves as navigational aid, thus making navigation easier.

Preferably, the index and the data objects themselves should be displayed in parallel to fully exploit the advantages of an index.
Generalized Application Structures

Generalized application structures are prototypical structures that you can use to compare your application with. These structures are well known and have their typical uses. Therefore, you also can try to "trim" your application into the direction on one of these structures, if you want to take advantage of certain characteristics.

The most common generalized application structures are:

- **One-screen application**: All processing is handled on one screen.
- **Sequence**: Processing is handled on a sequence of screens.
- **Hierarchy**: The application consists of a hierarchy of screens.
- **Network**: The screens are fully or partially interconnected.
- **Index Structure**: An index structure is a metastructure. It is characterized by a static index that is based on the application structure or a central data structure. The index is used for navigation within the application or data structure.
- **Stack**: A stack is a dynamic metastructure with restricted navigational options. Instead of providing a representation of the whole structure, as an index does, a stack displays only the path to the current screen or page. The path provides users with the context of the current page and can also be used as an efficient means for navigation backwards or up the hierarchy.
- **Queue**: A dynamic metastructure of fixed size where the entries are organized either on a "first in - first out" basis or according to a ranking scheme.

Rarely a real-world application is an exact copy of one of these structures. Often, application structures are combinations of several of these structures. For example, a typical master data application consists of an entry screen, below which a sequence of screens displays the different views; the view screens may again lead to detail view screens. At the end of processing, the user returns to the entry screen.

For a more detailed description of application structures see Application Structures and Metastructures for Applications in Part IV: Background Information!

Application Structure and Navigation

Based on the processing and display requirements of the subtasks and the processed data, you distribute the subtasks and data on screens and/or subscreens. This process defines the application structure and the flow of processing. The flow of processing comprises the so-called macro navigation and micro navigation within an application. Without being too strict, we define both terms as:

- **Macro Navigation (Navigation)**: Navigation between screens (screen changes, changes to other windows) and between subdivisions (areas) of a screen (which usually serve different purposes).
- **Micro Navigation**: The low-level navigation between screen elements (controls or parts within controls); we restrict this kind of navigation to single screens, if they are not subdivided, or to areas within a screen, if the screen is subdivided.

The application structure is created from the functional components of a task as they emerge from the subtasks and the data processed within them. In the beginning of the design phase these components are abstract units - at least for more complex tasks. During the refinement of the processing, these units may be further subdivided. As more and more entities are created, these units mutate into more concrete objects that correspond to real-world objects like screens, dialog windows etc. In some cases, however, the final representation may emerge at a rather late state. For instance, you have to decide between different design alternatives and want to test these with users in a prototype. Here it is sufficient to use the abstract units until the final presentations are known.
Navigation - What Is It?

For the following we use the term "navigation" as a synonym for macro navigation. Navigation refers to the global movements of users between screens and or screen areas. Users can navigate

- **within** a task (e.g. within the screens of the task of creating a debtor),
- **to other tasks** with return to the current task ("trips" -> users leave the current transaction for a "service transaction" and return to it after the subtask has been completed),
- **between** "complete" tasks (work flow, e.g. users process the the different stages of an requisition which is done in different transactions).

From a more technical or presentational point of view, navigation may involve:

- **Screen changes** taking place in **one window**, screen changes which require movement to **another primary or to a secondary window**. In the case of modal secondary windows (dialog boxes) users are automatically forced to do these movements,
- movements to **other areas** (e.g. frames) on the same screen,
- movements to **views** (e.g. by clicking tabs in a tabstrip).

In the case of screen changes or modal dialogs the movements are obvious (it may, however, not be obvious to users, which button to press or which menu item to select to get there). In other cases (amodal windows; a wrong click may destroy any cues) the movements may not be so obvious to users, or the users themselves have to activate screens or screen areas.

There are also methods to hide navigation from users by creating partially stable working environments (see Perceived Stability - Hidden Navigation).

How Navigation is Supported

Navigation may be automatic or user-initiated. In the latter case users usually initiate these movements by selecting a **function**. How this function is called, can be very different:

- **Pushbuttons, Menus, Toolbars**: Provide navigation to other screens (random access). These screens may appear in the same window or in new windows.
- **Tabstrips**: Provide navigation (random access) between views while maintaining a stable environmental frame (tabstrip).
- **Road maps**, structure maps etc.: These maps allow to combine random access with orientational aids and context information (e.g. via screen previews).
- **Paths, History lists**: These devices mostly support backwards navigation (path information).
- **Index Frame**: An index frame is a stable screen area which provides links to objects, object parts etc. for easy and fast navigation between them. An index frame resembles a tabstrip in function, but may have very different presentations. It also accommodates more flexible structures than a tabstrip: While the tabstrip just provides a linear random access structure, an index may be linear, a simple (two or three levels) hierarchy or even a complex hierarchical or network structure.

Which of these methods should be chosen in a certain situation, depends on a number of factors like general guidelines (e.g. the design framework), the accessed objects, the user group etc.

From the users’ point of view navigation has to be obvious and understandable, not like "magic". For more information on these aspects, see sections User-Oriented Design Principles!
Navigation and Interaction

As we already learnt, interaction and navigation are closely connected. The most common case is that users click a button for a navigation function or select one from a menu. In other cases users actively click other windows or screen areas to proceed with processing there (in some cases they return to their starting point).

Since the more widespread use of drag & drop operations a new combination of interaction and navigation comes into play: Users drag objects into other screen areas, windows or onto the desktop in order to initiate an action on them (the most famous drag & drop operation is dragging files into the trash can in order to delete them). Drag & drop operations are by no means obvious - despite their appealing resemblance to physical actions. And sometimes these operations turn into disasters when users inadvertently release the mouse button and when there is no undo function!

Automatic navigation where users "magically" are transferred to a certain screen after initiating an action is a "double-sided sword": In some cases users find it very helpful, in other cases they wonder how they ever got to a place and lose their sense of control.

Source: From Analysis to Design: Bridging the Gap
## Structure and Navigation - General Considerations: Addendum

Back to Structure and Navigation - General Considerations!

### Overview of the Criteria and Possible Values

**Dimensions: Structure - Formal Characteristics**

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Origin of Structure</td>
<td>Flow of Processing - Data Structure</td>
</tr>
<tr>
<td>Basic Task/Object Structure</td>
<td>none - linear (sequential) - hierarchical - complex</td>
</tr>
</tbody>
</table>

**Design Options: Structure**

<table>
<thead>
<tr>
<th>Option</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Application Structure</td>
<td>Simple One-screen application - Sequence - Hierarchy - Network - Compound Screen (Complex One-screen Application)</td>
</tr>
<tr>
<td>Compound Screens: Metastructure</td>
<td>None - Index Structure - Stack- Queue</td>
</tr>
<tr>
<td>Compound Screens: Screen Structure</td>
<td>No Division - Vertical Division - Horizontal Division - T-Structure - Complex Structure; Overview-Detail Structure</td>
</tr>
</tbody>
</table>

**Dimension: Navigation - Formal Characteristics**

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Macro Navigation - Micro Navigation</td>
</tr>
<tr>
<td>Range</td>
<td>Within Task - Trip (to other Task, temporary) - Between Tasks</td>
</tr>
</tbody>
</table>

**Design Options: Navigation**

<table>
<thead>
<tr>
<th>Option</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Movement</strong></td>
<td>Screen Change in one Window - Screen Change to another Primary or Secondary Window - Movement to other Area on the Same Screen - Movement to View (on the same Screen)</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Mechanism</strong></td>
<td>Drag b&amp; Drop - Pushbutton(s) - Menu - Toolbar - Tabstrip - (Road/Structure) Map - Path (Stack) - History List - Index - Link List</td>
</tr>
<tr>
<td><strong>Mode</strong></td>
<td>User-Initiated - Automatic</td>
</tr>
</tbody>
</table>

Back to [Structure and Navigation - General Considerations](#)!

**Source:** From Analysis to Design: Bridging the Gap
Structure and Navigation - How to Optimize It


Here we present the more "practical" aspects of application structure and navigation. We propose several methods that can help developers to optimize structure and flow of processing, that is macro and micro navigation, for their application.

For structuring an application and determining its macro navigation, we propose to sketch or model structure and navigation by using paper and pencil, a blackboard, or graphics tools. It is also useful to compare the application structure to generalized structures in order to profit from solutions that have already been found for these structures.

Micro navigation refers to the fine-grained navigation within one screen or screen area. Here, we propose to draw flow graphs to indicate and revise the flow of processing.

See Structure and Navigation - General Considerations for more general structural and navigational aspects of application design! See also Application Structures, and Metastructures for Applications for background information.

Macro Navigation (Navigation)

The flow of processing within a task consists of smaller steps that can be seen as micro navigation, and larger steps that can be regarded as macro navigation. Here we cover macro navigation. Micro navigation is handled below, because it belongs to the more "fine-grained" design issues.

(Macro) navigation usually means that users go to different screens. This often implies context changes for the users, because the new screens look different from the preceding ones. Users may also lose orientation, that is the sense of where they are, during navigation when orientational cues are missing. The importance of these cues for users feeling comfortable cannot be overstated. Current applications provide far too little context and orientational information!

The art of design seems to be balancing complexity with the orientational needs of users. No "general purpose" solutions can be provided here. But a good heuristic is, to keep everything as simple as possible...

Macro navigation may require a change in the input medium (e.g. from keyboard to mouse), though keyboard shortcuts may be helpful for power users. Macro navigation, too, influences the efficiency of an application: If the navigational steps are not evident or contrary to common conventions, users will have severe difficulties and be slowed down.

Perceived Stability - Hidden Navigation

Techniques that provide a stable and familiar working environment are a valuable design option to support user orientation. Usually this stability is achieved by keeping a stable frame of reference while exchanging certain screen areas (instead of changing the whole screen). Such techniques are, for example:

- **Tabstrip**: A rectangle subscreen within the tabstrip is exchanged. Similarly, buttons on the screen may cause the exchange of a fixed screen area (e.g. used with touchscreens).
- **Index layout**: Depending on the selected index item, a frame or subscreen is exchanged.

Keeping the context stable effects that users do not perceive the navigation as such. Thus navigational complexity is reduced. But there is also a drawback: These techniques require space and make screens more complex. For example, maps and graphs
of the application structure require additional windows or screen areas, and thus also increase visual complexity. In addition, often abstract graphs are used that users do not understand (they did not study computer science, like the developers did).

To sum up, these techniques often require more screen space, reduce the effective work area and lead to more complex screen layouts. Therefore, the expected user groups as well as task requirements should determine, whether you choose this design option.

Modeling Structure and Navigation

Having determined the basic application structure is a good reference point for checking the structure and the resulting navigation. To our opinion a "mental" check does not suffice, because an application structure and the navigational paths within it can become quite complex. Therefore, we propose to draw a sketch of the application structure and to link the structural elements through navigational paths. Annotate the links with the functions used, the direction of the link, restrictions etc. The linking need not be carried out in full detail, but the most important paths should be included. If needed, you can model the detailed navigation in "detail models" and thus keep the main model simple.

Uses

Modeling the structure and navigation of an application can be useful for:

- comparing different application structures in order to settle on the simplest and most transparent one,
- comparing the complexity of the navigation of different solutions in order to settle on the simplest one
- comparing different solutions with respect to the number of required steps for a procedure, screen changes, pages changes etc.

It is also useful to compare the structure of your application with generalized structures. Though your application's structure will usually be more complex, this comparison can help you in understanding your application and in transferring design solutions that already have been successfully implemented for these generalized structures.

Tips for Creating Models

Boxes and links are the basis elements in your models. Boxes may represent objects, tasks, screens, HTML pages or whatever you like. Connect the boxes with links to indicate structural relations and navigational paths.

You can sketch the models with paper and pencil or on a blackboard. You can also use PostIt!s for the boxes and fix them to a blackboard or flipchart. This makes it easier to rearrange the boxes (threads may be used for the links). Alternatively, you can use graphics tools for this purpose.

Label the boxes with useful information like screen or page title, purpose, objects (data), etc. Also label the links and indicate their directions and possible restrictions.

If the model has settled, it is useful to create a simple HTML prototype with links between the different screens or pages. This allows you to check the navigation between screens or pages in a web browser and already get a dynamic view of the navigation within your application.

Flow of Processing - Micro Navigation
The **flow of processing within a task** may consist of smaller steps that can be seen as micro navigation, and larger steps that can be regarded as macro navigation. **Micro navigation** to a large part determines the efficiency of an application.

- **Direction**: Micro navigation should follow the common heuristic of going from bottom to top, and from left to right. There should not be frequent or even erratic changes in direction.
- **Input device**: Micro navigation also should not require frequent changes between keyboard and mouse input. When such changes are necessary, consider providing keyboard shortcuts that allow users to stick to the keyboard. This is especially important when users have to enter many data. Here, for example, tab chains help users to efficiently enter data without having to leave the keyboard.

**The Flow Graph**

If you want to optimize micro navigation in your application, it might be a good idea to draw a **graph** that illustrates the basic flow of processing for your screens. This graph overlays the screens and consists of lines that indicate the direction on processing within and between screen areas.

Navigation between different screens may be represented by lines connecting pictures for different screens.

For example, data entry into a couple of fields may be represented by a line downwards. However, for saving the user has to press the “Save” button which requires a movement up to the system toolbar. During processing the user may be forced to do several of these “zig-zag” movements which is quite inefficient and unintuitive. So, if your screens require many zig-zag or circular movements, you should consider a redesign of your screens!

![Figure: An example for a flow graph](image-url)
Source: From Analysis to Design: Bridging the Gap
Users and Work Place

User Impact on Design | Work Place Impact on Design

We added users and work place to our list of dimensions, because both have characteristics that constrain the possible design space. For example, certain users may not be able to cope with complex applications, do not understand advanced interaction modes, or cannot use sophisticated controls. Work places may also have properties, like time pressure or high error risk, that exclude certain design options.

Overview of the Criteria and Possible Values

Dimension: Users

For users we build on the two basic criteria computer knowledge and domain knowledge. Somewhat orthogonal to these is the criterion frequency of use. Here we identify the casual user as an important user type with special requirements for the design.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Knowledge</td>
<td>Beginner - Intermediate - Proficient</td>
</tr>
<tr>
<td>Domain Knowledge</td>
<td>Little or none - Basic - Good</td>
</tr>
<tr>
<td>Frequency of Use</td>
<td>Regular - Casual</td>
</tr>
</tbody>
</table>

Dimension: Work Place

For business applications, usually work place characteristics are of lesser concern. However, as SAP is moving into new application areas, we will add a few comments on this issue, too.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning/Training</td>
<td>Possible - Not possible</td>
</tr>
<tr>
<td>Time Pressure</td>
<td>None - Low - Medium - High</td>
</tr>
<tr>
<td>Risk</td>
<td>None - Low - Medium - High</td>
</tr>
<tr>
<td>Type</td>
<td>Office - Mobile - Factory - Public system</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal - Rugged (different types)</td>
</tr>
</tbody>
</table>
**User Impact on Design**

The most basic criteria about users that developers need to consider are:

- Computer Knowledge: How **computer-literate** are the users?
- Domain Knowledge: How much do the users know about the **task domain**?

Both of these criteria influence design by limiting the possibilities for using controls and interaction modes. In the following table you can easily see what the most important design aspects for the prospective users of your application are. Note however, that there are exceptions to this general picture - see below!

<table>
<thead>
<tr>
<th></th>
<th>Beginner</th>
<th>Intermediate User</th>
<th>Proficient User</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Computer Knowledge</strong></td>
<td>Little or none. This means:</td>
<td>Intermediate to good. This means:</td>
<td>Good to excellent. This means:</td>
</tr>
<tr>
<td></td>
<td>- Knowledge of GUI conventions (controls, interaction) cannot be assumed.</td>
<td>- Knowledge of most used GUI conventions (controls, interaction) can be assumed.</td>
<td>- Knowledge of GUI conventions (controls, interaction) is present; knowledge of shortcuts, tricks etc. can be assumed.</td>
</tr>
<tr>
<td></td>
<td>- Use simple and basic controls and interaction only. Example: Use a point &amp; click interface with pushbuttons.</td>
<td>- Do not use sophisticated controls or interaction modes.</td>
<td>- You can use sophisticated and efficient interaction modes and controls.</td>
</tr>
<tr>
<td><strong>Little or no Domain Knowledge</strong></td>
<td>Design intuitive screens with</td>
<td>Like beginner, but</td>
<td>State-of-the-art GUI, but users need</td>
</tr>
<tr>
<td></td>
<td>- only basic controls,</td>
<td>- more advanced controls can be used.</td>
<td>- onscreen instructions or easy to access help (concise!),</td>
</tr>
<tr>
<td></td>
<td>- little or no complexity (at best full screen),</td>
<td></td>
<td>- background information - should be available, but can be more &quot;remote&quot;,</td>
</tr>
<tr>
<td></td>
<td>- onscreen instructions or easy to access helps (concise!),</td>
<td></td>
<td>- affordances, diagrams, ...</td>
</tr>
<tr>
<td></td>
<td>- affordances, diagrams, ...</td>
<td></td>
<td>- context information,</td>
</tr>
<tr>
<td></td>
<td>- no or only simple navigation</td>
<td></td>
<td>- orientational aids.</td>
</tr>
<tr>
<td></td>
<td>- context information,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- orientational aids,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- simple flow of control on screen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic Domain Knowledge</td>
<td>Like beginner, but</td>
<td>Like beginner with basic domain knowledge, but</td>
<td>State-of-the-art GUI, with the following features:</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------</td>
<td>--------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>help can be reduced to provide more screen space, affordances are still useful, to guide the flow of processing.</td>
<td>more advanced controls can be used.</td>
<td>Onscreen instructions or easy to access help should still be available, but can be reduced.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Background information should be available, but can be more &quot;remote&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Affordances, diagrams, may still be useful, but can be sparse.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Personalization is useful.</td>
</tr>
<tr>
<td>Good Domain Knowledge</td>
<td>Like beginner, but</td>
<td>Like beginner with good domain knowledge, but</td>
<td>State-of-the-art GUI can be used.</td>
</tr>
<tr>
<td></td>
<td>help can be further reduced to provide more screen space and less distraction, affordances are still useful, to guide the flow of processing.</td>
<td>more advanced controls can be used.</td>
<td>Onscreen instructions or affordances, diagrams, ... are usually not needed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Help and background information can be more more &quot;remote&quot;, but should still be available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Screen space can be used for maximum efficiency (including shortcuts, etc.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Personalization is useful.</td>
</tr>
<tr>
<td>Consequences</td>
<td>Only a restricted set of controls and interaction modes can be used.</td>
<td>Like beginners, but the interface need not be simplified, because basic GUI knowledge is present.</td>
<td>State-of-the-art GUI can be used. Efficiency is the primary issue here.</td>
</tr>
<tr>
<td></td>
<td>Needed domain knowledge support depends on domain knowledge.</td>
<td>However, the interface should not be too sophisticated and still be &quot;straightforward&quot;.</td>
<td>Needed system support for domain knowledge depends on degree of domain knowledge.</td>
</tr>
<tr>
<td></td>
<td>On all levels of domain knowledge, the application should be self-explaining and support the users´ needs for orientation and context.</td>
<td></td>
<td>Background information should be available, but can be &quot;hidden&quot;.</td>
</tr>
<tr>
<td></td>
<td>Personalization is not needed.</td>
<td></td>
<td>System support for actual interaction can be kept minimal for more efficient use of screen space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Personalization is useful.</td>
</tr>
</tbody>
</table>
Exception to the "Basic Rules" - Casual Users

There is another important criterion, the "**frequency of use**". While users uses most applications on a **regular** basis, there are some applications that they use only **casually**. While regular users are the users most developers have in mind, casual users are special, because they may have any degree of computer literacy and of domain knowledge, but do certain tasks only **occasionally** - maybe just once a year. So even if they may possess good domain and computer knowledge, they may not know how to operate a seldom used application. In our table casual users come closest to the cells "beginners/intermediate" combined with "little or no domain knowledge", because they require intuitive, easy to operate screens. Casual users also need **consistency** for better information retention and context as well as orientational information to feel secure.

Work Place Impact on Design

Business Work Places

For **business applications**, usually work place characteristics are of **minor concern**. Neither do they have special environmental conditions, nor are users under extreme time pressure, or are errors of primary concern.

However, there may be work places where users have **little or even no time for learning** an application. In other companies, user fluctuation may be so high that extensive training is not economical and therefore not provided. These work places require simple and intuitive applications that can be handled by "everybody" and without prior training.

Special Work Places

Some work places are "special" because they differ in their requirements from usual work places. Here are some examples (together with the respective criterion):

- **Time Pressure**: Work places with high **time pressure** (e.g. phone operators),
- **Risk**: Work places with **high risk**, where errors are dangerous (e.g. nuclear power station, hospitals, ...),
- **Type**: Public information systems, where users may have no prior computer knowledge or training (kiosk systems in museums or department stores, railway station info systems, ...),
- **Environment**: Rugged work places (harsh environmental conditions) (e.g. fork-list truck driver, systems in chemical plants, mobile systems in airplanes or on the sea, ...).

Special work places often require a simple, intuitive and efficient design independent of the users' degree of computer and domain knowledge. In some cases users have little or no computer knowledge. In other cases error prevention is a critical issue.

Most of these special work places are not served well by a standard GUI - even though many software designers try to do just this. These work places usually comprise just one or a few tasks. Therefore, they do not need a full-fledged GUI with movable windows, trees, sliders, and what else is at your disposal. Most of them are just need a simple full-screen layout with a point & click interface. Whenever you design a specialized application, ask yourself whether you want your VCR, washing machine or photocopier to run Windows 2000.

Source: From Analysis to Design: Bridging the Gap
Presentation of Data

Presentation of Data Sets | Presentation of Single Data | Text vs. Graphics | Presentation and Interaction

After determining the basic views of the data and their interactions, the actual presentation of the data has to be decided. Below we collect some considerations on presenting data sets and singular data objects. Note that presentation and interaction are closely related: Most presentations clearly determine how users can manipulate objects or object attributes.

Overview of the Criteria and Possible Values for Dimension Data

For data we identify the basic criteria data set structure, attribute presentation, alternative data presentation, and format:

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Set Structure</td>
<td>None - Unordered set - List - Simple Hierarchy, Hierarchy, Network</td>
</tr>
<tr>
<td>Attribute Presentation (Internal Structure)</td>
<td>Field(s) - Single table - Parallel table - Tree</td>
</tr>
<tr>
<td>Alternative Data Presentation</td>
<td>Single - Few - Many</td>
</tr>
<tr>
<td>Format</td>
<td>Text - Graphics - Mixed</td>
</tr>
<tr>
<td>User: Domain Knowledge</td>
<td>Beginner - Intermediate - Proficient</td>
</tr>
<tr>
<td>Workplace Type</td>
<td>Office - Mobile - Factory - Public system</td>
</tr>
</tbody>
</table>

For the user and work place criteria see Users and Work Place!

Presentation of Data Sets

The following table lists some basic guidelines for presenting data sets in a text-based or semi-graphic fashion:

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Presentations</th>
<th>Alternative Presentations</th>
<th>Do Not Use!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unordered Set</td>
<td>List, table, graphical list</td>
<td>Map (image map), graphical clusters</td>
<td>Tree</td>
</tr>
</tbody>
</table>
### Presentation of Data

<table>
<thead>
<tr>
<th>List (Ordered Set)</th>
<th>List, table, graphical list</th>
<th>Map (image map), graphical clusters</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Hierarchy (2 to 3 Levels)</td>
<td>Two- or three-level list or table; graphical lists with header</td>
<td>Graphical clusters with &quot;header objects&quot;</td>
<td>Tree</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Tree, tree graph</td>
<td>Hierarchical lists, tree diagrams (best like in outliners with the ability to expand and collapse partial structures)</td>
<td>Long linear lists (if there are many objects)</td>
</tr>
<tr>
<td>Network</td>
<td>Tree with interconnections, network graph</td>
<td>Hierarchical lists, network or tree diagrams with interconnections (best like in outliners with the ability to expand and collapse partial structures)</td>
<td>Long linear lists (if there are many objects); for few fully interconnected objects a list which allows random access is OK.</td>
</tr>
</tbody>
</table>

### Legend

- **Presentation**: Presentations that are typically used
- **Alternative Presentations**: Graphical presentations, special cases
- **Do Not Use**: Presentations that should not be used

Text-based presentation means that the objects are displayed as texts, namely as labels or numeric values. However, most of these presentation have some "analog" or graphical dimension as well - that is why we call the presentations "semi-graphic":

- **Lists** are written in a linear order from top to bottom or left to right (or vice versa),
- **tree** presentations somehow reflect the tree structure in their layout,
- **sets** are usually presented in a list-like style
- **networks** are harder to display as text structures, but in some cases they can be visualized as lists or trees as well.

See below for more on the issue of text vs. graphics!

### A Comment on Trees

Currently using index layouts with trees (the so-called "Explorer paradigm") seems to indicate that your user interface is "up-to-date". However, up-to-date does not mean "user-friendly"! Trees are visually complex and require good mouse abilities. Also, even their widespread use in Windows does not guarantee, that every user will know how to use it.
Apart from being "fashionable", there is another characteristic that attracts developers, a tree is "generic". If you do not know in advance which structure the data set will have, you simply put a tree on the screen and you are done! This way you put the presentational burden on the users' shoulders. A generic solution is the optimal solution only from a technical point of view, not from the users'!

Look at the web, and you will find many alternative solutions to using "generic" trees, especially if the number of the data in a set is fixed or small!

**Alternatives to Trees**

The number one option for simplifying trees is to reduce the hierarchy to a flat or two-level list:

1. **Reduce tree to flat list.**
   This option is useful,
   - if the number of items is relatively small, and
   - if the hierarchy is artificial to users (do not present internal structures to users if they do not need or understand them!)  
   - if an ordering scheme can be imposed on the data that helps users to easily find data (the simplest order is an alphabetic one)

2. **Reduce tree to two-level list.**
   This option is useful
   - if the number of items is larger than in case 1
   - if there are “natural” categories that users understand

**Some Don'ts for Trees**

- Do not use trees for simple lists with a header, e.g. where the root node acts as header and where the leaves act as items!
- Do not use trees for simple tables (where the root might again act as a header)!
- Do not use trees for two-level lists, e.g. the first-level nodes as category names, and the second-level nodes as items!

If there are good reasons for using trees in these cases, consider turning lines off and using different font sizes for different levels to improve visual grouping. Trees are very poor in separating different hierarchical levels as is needed, e.g., in tables of contents or simple two-level lists, where items are sorted by category.

**A Comment on Lists and Clusters**

**Lists** come in many disguises, especially lists with few objects. The vertical or horizontal textual lists are just the most basic presentations. Graphical lists can have so many different looks, far too many to be listed here! For instance:

- Horizontal or vertical lists of icons/graphics for objects or functions,
- Images displaying objects or having objects or object parts as parts (e.g. the picture of a bureau or a machine),
- Maps, e.g. a road map where items might be cities,
- Calendars for time-based information.

Lists can also have a hierarchical structure, but should not have more than two to three levels.

**Clusters** are different from lists in that objects may have no recognizable order or may even be rearranged by users. Like lists clusters may have a hierarchical structure. For example a set of "daisy blossoms" might present a two-level cluster.

**Presentation of Single Data**

Data presentation is too big an issue to be covered in detail here. Therefore, we present only some general considerations.
Presentation of Data

General Presentation Rule

Always try to implement the presentation that is the most efficient and understandable one for a given task and user population.

Basic "Aggregate" Presentations

In R/3 and similar business software systems there are the following basic modes for presenting objects:

- **Template View (Field View):** All or most attributes of an object are displayed on a template. Usually these attributes are presented as input/output fields.
- **Single Table View:** A selection of attributes is displayed in a table with the attributes in rows. The remainder of the attributes may be displayed in a detail view.
- **Parallel Table View:** Usually a selection of attributes is displayed. Several objects are displayed in parallel. Typically objects are arranged in rows and attributes in columns. For few objects with many attributes, the order should be reversed.
- **Tree View:** Objects with more complex inner structure may be presented as trees or tree-like lists.

"Atomic" Presentations: Display of Names, Attributes and Values

On the basic level, data are usually **names** or **numbers**. A name or number may characterize an object as such or one of its attributes. Thus on an "atomic" level the question is how names and numbers can be displayed. The usual answer to this is use a text or number string that is displayed in a field or an aggregate of fields like a list, table or tree. But many attributes have more "natural presentations" than these, like images, simulated analog devices, and the like:

- Alternative presentations to single or few fields:
  - Radiobuttons, checkboxes, spinbuttons, sliders, progress indicators, gauges, status icons, images
- Alternative presentations to field groups, lists and tables:
  - tables or lists of status icons, charts, images, diagrams, maps, calendars, ...

Text vs. Graphics

Currently business applications are dominated by text-based presentations, because from a programmer's point of view these are easier to handle, more general (or "generic"), and often use less space than graphics. In the future, however, the use of graphics will increase, because graphics make better use of the abilities of our visual perception:

- they help users to understand "what the data say"
- they support quick scanning for the relevant data
- they can visualize relations that are difficult to find in textual presentations

This does not mean, however, that text-based presentations are no longer needed. Text-based presentations are exact (digital), while graphical presentations are imprecise (analog). Users always need to go back to the text-based presentation, when they want to know the exact numbers. Graphics, however, help to get a quick overview and a feeling for the relevant things.

The borderline between text-based and graphical presentations is very fluid:

- Absolute numbers or percentages can be converted to areas, sticks, blocks or even images of the items themselves and then inserted into the basic "text-based" structure.
- On the other hand, a set, list, tree, or network can be displayed graphically, but the items can be displayed as text labels or numbers.
- And, of course, if appropriate, the whole structure can be displayed in full graphical style. Often, however, it helps to combine
Presentation of Data

both approaches and to supplement a graphical presentation with text labels.

For more information on the issue of text vs. graphics and on the correct use of pictures and charts, please consult the literature (see some suggestions!)

Presentation and Interaction

Presentation of data is closely connected to possible interaction with the data: Choosing a presentation also largely determines which interaction is possible with the data and which not. With input/output fields, interaction is very basic: Users can edit text, may cut and paste or even drag and drop the text or parts of it - but that's it!

Other presentations like radiobuttons, checkboxes or spin buttons require simple mouse clicks (sometimes keyboard interaction is possible, too) or mouse dragging (sliders, gauges, ...). More advanced graphical presentations may allow users to interact with the data in even more “natural” and direct ways.

For special users and/or work places it may be necessary to restrict interaction which in turn influences data presentation. On touchscreens, for instance, a point & click interface is preferable, while dragging objects should be avoided. Thus sliders should not be used in this case.

Selected References


William Horton (1994). Designing and writing online documentation. New York: Wiley. (See chapter pictures!)


Source: From Analysis to Design: Bridging the Gap
Presentation of Functionality

Types of Functionality | Presentation and Access of Functionality | With all these Possibilities - Where Are the Problems?

Functionality is usually not discussed under the "presentation" label. But this view helps us to take a user perspective on this topic and clarify why users have problems with finding, accessing and using functionality.

Overview of the Criteria and Possible Values for Dimension Functionality

For function we identify the three basic criteria type, access method and format:

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Task-related - Interaction-related</td>
</tr>
<tr>
<td>Access Method</td>
<td>Mouse - Keyboard - Touchscreen (with or without pen)</td>
</tr>
<tr>
<td>Format</td>
<td>Text - Graphic - Mixed</td>
</tr>
<tr>
<td>User: Computer Knowledge</td>
<td>Beginner - Intermediate - Proficient</td>
</tr>
<tr>
<td>Workplace Type</td>
<td>Office - Mobile - Factory - Public system</td>
</tr>
</tbody>
</table>

For the user and work place criteria see Users and Work Place!

Types of Functionality

Most applications offer too much functionality for presenting all their functions on the screen. Also "all functionality is not the same": There is task-related functionality, and there is also a lot of functionality that is not directly task-related but serves for interaction purposes. Examples for task-related functions are functions to create or edit different objects in the R/3 system. Examples for the latter are the clipboard functions or table-related functions like sorting and filtering.

Task-Related Functionality

Specific vs. Generic Functionality

Task-related functionality is functionality that is directly related to the task. This functionality may, however, have generic character and may recur in many other applications. For example, the typical R/3 functions for creating, editing and displaying objects are such generic functions. Other task-related functions are specific to the task at hand.

Priority: Primary vs. Secondary Functionality
Presentation of Functionality

Primary functionality is the functionality that is absolutely necessary to perform a certain task. Without calling these functions, the purpose of an application cannot be achieved. Secondary functionality is optional; it is not necessary, but may be very useful, e.g. for

- achieving variations of the results
- achieving results in different ways (through different steps or procedures)
- doing certain (sub)tasks faster and more efficient

For secondary functionality it is much more difficult to decide, which should be included, which should not, who needs or uses it, and how it should be presented. Opinions on frequency of use, usefulness etc. vary within the development teams and may lead to long and frustrating discussions.

Interaction-Related Functionality

Interaction-related functionality may be difficult to separate from task-related. As a rule of thumb we subsume here

- functions for interacting with tables, tree and other complex controls
- functions for interacting with windows like scroll functions, close, move etc.
- clipboard functionality like cut, copy and paste
- drag & drop functionality and other functionality which is implicit in the interface

Interaction-related functionality is widely standardized and common to all applications. It is, however, based on conventions like any other activities of our daily life and by no means evident to beginners or people who do not use computers. It also may be surprising to many developers to realize how much of this functionality is included even in the simplest application!

Task-Related vs. Interaction-Related Functionality

Ideally an application should only have task-related functionality and none or only a minimum of interaction-related functionality. This, however is not possible with today's software and computers. But prevent interaction-related functionality from dominating the screen, which makes it difficult or even impossible for users to find the task-related functions.

Though there are user interface standards for assigning functions to menus and arranging them in toolbars or on the screen, often there is a confusing mixture of both types of functionality that puzzles users and makes it hard for them to find the important functionality.

Moreover most screens boast with implicit or hidden functionality which makes it even harder for users to interact with the application. Much of this functionality is well known to users who are computer-literate, because is adheres to the conventions of graphical user interfaces, but for beginners it is hard or even impossible to "do want they want to do". Remember, for example, the secretary who wants to do the same things with her new word processor that she successfully with her typewriter!

Presentation and Access of Functionality

Presentation of functionality and access to it are closely related, because the presentation to a large amount also determines how the functions are accessed - at least with the mouse.

Basic Access "Standards"

Presentation of functionality is fairly standardized in graphical user interfaces:


**Presentation of Functionality**

- **Pulldown menus (or similar types):** The functionality is too much to be displayed on the screen; therefore it is assigned to standardized or self-defined categories. Only the categories are permanently visible on the screen - usually in a reserved area - while the subcommands are displayed on demand.

- **Onscreen menus:** The command set is small enough to be fully displayed; this is usually done in the work area or on special "menu screens"; for larger command sets onscreen menus may be organized in hierarchies with a hierarchy of menu screens.

- **Toolbars:** Important functions are (usually in addition to being assigned to the menus) arranged in toolbars; toolbars can be customizable.

- **Floating toolbars:** A variant of the toolbars that always stays on top and can be moved around or minimized by the user.

- **Onscreen buttons:** Function buttons that are located in the work area, preferably close to the objects they affect.

- **Keyboard commands:** May be shortcuts in addition to mouse access mechanisms or entered as text commands (usually in a special "command line" or field).

There are also some "newer" access mechanisms:

- **Links (textual, graphical):** With the advent of the web, links have become an option for navigational functions.

- **"Sensitive" areas, maps:** Here functions are evoked by clicking a certain area on an object; this feature may be more or less evident to users.

- **Drag & drop:** This is a special variant of a mouse command; users drag selected objects onto other objects in order to evoke an action.

- **Voice commands:** Voice commands usually are an addition to the default access mechanisms in order to simplify function access for certain uses.

**Keyboard Access vs. Mouse Access**

The following table lists the accessibility of functions for mouse and keyboard commands:

<table>
<thead>
<tr>
<th>Interaction Element</th>
<th>Mouse Access</th>
<th>Keyboard Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pulldown Menu</td>
<td>Click in option</td>
<td>Accelerator key or key code (e.g. Ctrl-X)</td>
</tr>
<tr>
<td>Onscreen Menu</td>
<td>Click in option (or button)</td>
<td>Enter the option</td>
</tr>
<tr>
<td>Toolbar</td>
<td>Mouse click</td>
<td>Usually only, if the function also appears in the menu</td>
</tr>
<tr>
<td>Floating Toolbar</td>
<td>Mouse click</td>
<td>Usually only, if the function also appears in the menu</td>
</tr>
<tr>
<td>Onscreen Button</td>
<td>Mouse click</td>
<td>Accelerator key; if the function is also in the menu through the respective command</td>
</tr>
<tr>
<td>Link</td>
<td>Mouse click</td>
<td>---</td>
</tr>
<tr>
<td>Sensitive Area, Map</td>
<td>Mouse click</td>
<td>---</td>
</tr>
</tbody>
</table>
Voice Command | Only, if the function also accessible through the mouse | Only, if the function also appears in the menu

**Note:** In Windows all interface elements can be accessed through the keyboard by moving the focus to the respective element; however this access is often more cumbersome than keyboard shortcuts.

**Users: Mouse or Keyboard?**

There have been long debates about which is better - mouse or keyboard. Today there is little doubt that both mechanisms are useful and have to be supported. Keyboard access is also a legal requirement to make computers accessible to blind users.

When to use which mechanism usually depends on frequency of use and user expertise. While mouse access mechanism are fairly evident to most users they are slower and may cause arm strain for repeated actions. Keyboard access comes into play, when commands are used with high frequency and repeatedly, when users want to work fast, or when they do not want to move their hands from the keyboard.

As it takes time to learn keyboard commands, most users know only a couple of keyboard shortcuts. This usually changes as soon as the application usage changes, and when it pays to learn more shortcuts.

**Text vs. Graphics**

There abound masses of literature about the pros and cons of textual vs. graphical presentations. So, we do not want to reiterate that. As a matter of fact, for long text-based presentations have dominated business applications. A recent addition are diagrams for displaying tabular data. However, this feature is not a "revolution" but only an addition to the existing presentations. What is really needed is a perfect integration of text, graphics and other presentation forms. This, however, requires a lot of work and knowledge about presentation forms on the developers side - if it is technically feasible at all. Also the best presentation for a given situation depends very much on the context of the data which may not be exactly known to the developer.

Given all this - including the technical restrictions - it is no wonder that developers preferred and still prefer the generic text-based presentations. However from the users' point of view this state of affairs is unsatisfactory, and it is highly desirable that data presentation becomes more flexible and much more graphics-based.

**With all these Possibilities - Where Are the Problems?**

**Complexity**

The most severe problem with functionality is its sheer mass that is found in many applications. As screen space is limited, functionality has to be classified, organized into sets or hierarchies and partially hidden from view.

Menu guidelines provide some guidance about where to place functions within menu tree, however, they neither solve the complexity nor the availability problem (see below). These guidelines are "logical" and "mechanistic" - not to say "technocratic" - they care little for users: They assume that users understand the basic menu structure and are able to search it efficiently for hidden commands.

**Warning:** Cascading menus are a perfect way to hide functionality - especially, if they have more than two levels!

The most efficient way out of the complexity trap is to design simple applications with less functionality. We even do not want to
propose in this guide schemes that tell how masses of functionality can be organized - this would undermine any simplification attempt.

**Availability**

Availability is a psychological term. It reminds us that users only use the functionality that they are aware of. They are most aware of the functionality that is displayed on the screen, especially the one that is displayed in-place, that is close to the objects that it affects. But, see below, putting functionality on the screen does not guarantee that users find it.

Users may also be aware of hidden functionality, but this requires that they are proficient or at least domain experts. In the latter case, they may know that there should be a certain functionality, but they may not be able to find it.

**Background - Foreground**

This also a psychological term dating back to the Gestalt laws. It states that there is information that we are aware of, that is “signal” to us, and that there is also information that we are are not aware of, that is “noise” or background to us. For user interfaces it is a problem, if necessary functionality is dominated by optional one and made “background” information. In this case, even though the functionality is visible, users may not find it - it is simply not available to them.

**Prioritizing and Optimizing**

There is no solution to the functionality problem that makes everybody happy. You have to prioritize your application's functionality with respect to what is really necessary and what is of secondary importance or can be totally left out. You also have to optimize the presentation as well as the access to the primary functionality, while restricting visibility of secondary functionality and making access to it harder. Optimization means overall improvement - not that everything gets better: The important and frequently used things become easier, while the others become more difficult.

**Source:** From Analysis to Design: Bridging the Gap
Refinement and Check

Refinement: Screen Layout Basics | Check of Suitability of the Design | Implementation | Connecting to the Other Side of the Gap: User Workshops and Usability Tests

The final design stages comprise the fine-grained screen design and checks for the validity of the design. The latter again connect to activities that let users participate in the design process, namely user days and usability test: Users verify the design with prototypes (early testing), comment on the application (user workshops) and test it (usability test).

Refinement: Screen Layout Basics

Finally, you refine your design by laying out the screens in detail. Here you assign controls to the screen, but it it still not necessary to have a "polished" design at this stage. Primarily the sequence of the fields and the representation of special fields should be decided here. Such decisions comprise:

- Use of controls for selections (radiobuttons, checkboxes, drop-down lists, ...) and for "special" fields (spin buttons, sliders, ...),
- placement of pushbuttons for global and local (object-related) functionality,
- grouping of related fields, naming of groups,
- use of certain interaction modes like drag & drop, point & click, etc.

More information on these issues are found in style guides like the SAP Style Guide for R/3, the Windows Style Guide, Web Style Guides, etc. The Design Guild contains the compact version of the R/3 Style Guide.

The actual "polishing" of the screen layout can be deferred until shortly before the application undergoes usability testing.

Check for Suitability of the Design

Now it is time to check the validity of the design! This check can be done with a more detailed paper or HTML prototype or even with a sequence of R/3 screens. Users should be able to simulate real-world tasks (that are based on the scenarios established in the analysis phase) with this prototype, but the actual functionality need not be present.

Utilize this check to implement changes to the design that improve its usability!

Typical questions are:

- Do users understand the screens?
- Especially, do they know how to accomplish the task there?
- Are users in principle able to accomplish the task?
- Where do users have problems with the design?

At this stage the focus is on conceptual issues, not on performance. Efficiency cannot be verified at this early stage, because functionality is not implemented sufficiently. However, some insight on this issue may already be gained. User performance should be tested later with the application itself or with advanced prototypes.
Refinement and Check

For more information on prototypes, see the section on **prototyping** in the Design Guild!

**Implementation**

After these tests have been conducted successfully, **implementation** can start. During the implementation phase it may still be useful to have more advanced prototypes at your disposal to repeatedly **verify the design**.

**Connecting to the Other Side of the Gap: User Workshops and Usability Tests**

**User days** in the form of workshops or usability tests can also help to check, whether design and implementation are still on track. Another method is the **usability review** which is conducted without user participation and on an "expert" basis only.

When the implementation is coming to an end, it is time for usability tests. Note, however, that the final usability tests in the lab might be too late for substantial changes to the application! Therefore, it is important to utilize the whole implementation phase for refining the design.

For more information on **user days**, **usability tests** and **usability reviews**, see the Design Guild!

**"Reusable" Scenarios**

**User roles** or **personas** and **scenarios** are a valuable and reusable tool for all these user contacts. They help to derive **real-world tasks** that can be used for testing **prototypes**, for testing the **application**, for conducting **reviews** as well as for design discussions.

**Design Alternatives**

When discussing the design (e.g. in design sessions) it is helpful not to stick to one design, but to discuss **alternative solutions** and to evaluate their **pros** and **cons**. Tasks or usage situations derived from scenarios are a good basis for weighting the pros and cons of design solutions and for finding criteria based on which an optimal design solution can be chosen.

Of course limited resources prohibit that design alternatives can be developed very far. But for the first conceptual stages, where paper prototypes or simple HTML prototypes can be used, we - contrary to Cooper - recommend a multiple approach, because it makes design decisions more explicit and prevents good ideas from being discarded, just because a hot debate took a certain route or because certain people took part in the discussions while others did not.

**Source:** *From Analysis to Design: Bridging the Gap*
Design guidelines assume an ideal development process: users have been investigated, realistic usage scenarios have been created, the developers listen to the user interface designers and eagerly implement their designs, the test results are satisfying. In real life, however, things do not work that smoothly. There are a number of stumbling blocks to be overcome, and lots of reasons why design projects may fail. In the following article, we present a couple of such issues based on past experience.

### Time Issues

#### No Time - And How You Can Make up for It

The most frequent excuse for bad design is lack of time: There was so much other work to do, and the schedules were so tight, that no time was left for going on site visits, creating scenarios, asking for usability consulting, reading usability guidelines, visiting usability-related Websites, or whatever the development team could have done to improve the design and usability of the application.

Inevitably, time schedules are very tight these days, but this is something we usually can’t change. So what can realistically be done under time pressure? At least the bare minimum can be done: The development team can - assisted by a user interface designer - conduct a 1-2 day workshop to

- Create usage scenarios and user roles (or "personas")
- Create simple paper prototypes for a first design proposal
- Do a quick-and-dirty evaluation of the design proposals using the scenarios; that is, let users perform real world tasks with the paper prototypes

If a quick evaluation of the prototypes cannot be carried out during the workshop, we strongly recommend demonstrating them at least to people outside of the development team. Ideally, the test persons should be prospective users, but coworkers from other departments, relatives, friends etc. also work out fine. It is very important to complete these activities within a few days and not distribute them over several weeks. Otherwise, the team will indeed run into timing problems. It may seem that there is no time to conduct such a workshop, but in reality it saves the team a lot of time which would have to be spent later fixing the design.

#### No Big Changes, Please - The Dilemma of Last-Minute Consulting

Imagine a typical situation in the daily life of a user interface designer. The designer is asked to have a "quick look" at a developer's screen designs. Often, however, it is not the problem the interface designer was asked to look at that makes him shake his head in quiet frustration. Instead, the application design is flawed as a whole, and the designer comes to the conclusion that a redesign would be necessary to solve the inherent problems of the application. Such a move is usually impossible to push through and also wasn't even the developer's original intention, which was just to fix a "small problem." There seems to be no way to prevent many big usability problems from making their way into the final product.

This situation becomes even worse when the release date creeps up and "help" calls to user interface designers increase in frequency dramatically. Usually, in such "last-minute consultations," the developer has a funny feeling about the design and calls the UI designer for a quick approval of what he or she has implemented. Or, the developer has a very tricky problem that he or she cannot solve.

While in both cases the user interface designer often finds a fundamental design flaw, all of his or her proposals have to be very simple changes to be accepted by the developer because there is hardly any time left (development closes "in an hour," or...
“tonight”). All major changes, which would substantially improve the application, are impossible to implement under the given time constraints. So, we end up with a frustrated user interface designer who, for example, only managed to reposition of a menu item and with a frustrated developer who hears for the first time, right before development close, that the design is fundamentally flawed.

Meetings

Frequency of Meetings

After the initial design phase, further consultations may continue over a longer period of time, maybe for three months. During this time there will - hopefully - be further meetings to refine the design, discuss the progress of the implemented application, and fix the problems with it. But how often should you meet? For usability consultants outside the development team the simple answer is, “the more often, the better.” As long as there is no evidence that the design is really implemented, you should meet at least once a week, in even more often. If meetings are delayed or canceled and the next meeting takes place only after three or more weeks, you may very well find that what has been implemented has little in common with the design ideas created during the last meeting. As we show below, it can be very hard to move the design back onto the right track.

Requirements Appear "Out of the Box" - Ill-Defined Scenarios, Meetings with Ever-Changing Participants

One of the most frustrating moments during a design discussion is when, after a long and heated debate, the team finally settles on a design and then all of a sudden someone pulls out a new requirement and asks “Have you thought of XYZ?” Of course, nobody has thought of it; all the other team members assumed that the requirements for the design were evident and laid down in the existing scenario descriptions. So, if the team agrees that the new requirement is important and should be included in the scenarios, that means that they were obviously not fully specified from the beginning. The problem with such ”late insights“ is that they often make the whole design crumble. The team may be forced to start over, loosing valuable time. Therefore, it is critical that scenarios are as complete as possible before designing begins. You can only design if you have a firm foundation. Any change in the prerequisites may lead to very different design decisions. But please, don't get me wrong, this doesn't mean that you should do without scenarios to avoid such problems. Even an ill-defined scenario is better than none.

There is another cause for "out of the box" requirements, namely design meetings. These meeting are often strung out over weeks and months with participants changing from meeting to meeting - sometimes with little or even no overlap between people. New people do not know much about the old design ideas and the rationale behind them. They tend to bring in new requirements and new design ideas - they have to justify their participation anyway, and the design will never stabilize.

Meet with the Right People

As experience shows, it is important to meet with the right people when consulting an application. If you meet with people who are willing to implement the design, be sure that they really are the people who implement it. Otherwise, some other developer may all of a sudden come up with an implementation that typically has little in common with the design that you and your discussion partners established during the meetings. And as we all know, a design that is already implemented typically wins.

The "Art" of Design

Random Walk Design

Often design discussions follow a “random walk” structure. The discussion starts with some initial design proposal, which is put forward by one or more team members, and then other members come up with their requirements and proposals. Typically, during such a discussion the design changes continually, depending on people's influence or loudness and the duration of the meeting, among other factors. Team members change opinions now and then, older designs “reappear” from time to time. After a couple of hours the group ends up with a totally different design - often achieved because most participants are too exhausted to
oppose the current proposal. This may not be bad in itself, if the design is much better than the old one. But usually the resulting design is just a product of chance: the rationales behind other designs, their pros and cons - if ever evaluated - have been completely forgotten. As a result of such erratic design discussions, the discussion starts anew with every follow-up meeting (usually it also has a couple of new participants). There, new proposals are brought in, old ones as well as their pros and cons are forgotten, and so on, and so on...

Coffee Corner Design

Many design meetings take place in coffee corners. An informal meeting culture is good for a company; it presents, however, a problem, when designs are discussed. In coffee corners are coffee machines, but there are no computers, whiteboards, beamers, flip charts or whatever devices that support the team members' imagination. Instead, as developers are used to abstract thinking and like it very much, design issues are discussed only verbally - maybe someone makes some cryptic scratches on a napkin. Hopefully, there are no logical errors in all the assumptions and design decisions made, but nobody really knows how the design looks or how its elements work together - and nobody seems to care. In our experience, it takes a lot of effort to force a team into a room with at least a whiteboard or at the computer and have a look at what they just "designed." But if you succeed, some decisions take minutes that before took hours.

Design by Authority

There are several types of authorities in interface design. For instance, a user interface designer is an authority for interface design because he or she has a lot of knowledge and experience in this field. In spite of this, there are cases, where developers or managers do not respect this authority; they believe that interface design is like raising kids or driving cars - it is a topic where everybody feels like an expert. Sometimes, however, it is the other way round. Developers leave all the design decisions up to the interface designer saying, "Tell me how to do it. I'll implement whatever you want." This is not the right way, either. Now developers are cut off from the decision-making process and if the design fails - even if they implemented something quite different - might put all blame on the interface designer in the end.

Another source of authority is managers. Sometimes managers want to take part in design meetings, for example, in order to "speed up" the discussions. Of course, the manager takes an active part in the discussion - he or she is the boss. As the discussion moves along, the manager may create a lot of clever design ideas. As a matter of fact, most of these ideas have already been discussed and discarded for one reason or another in previous meetings, but now it is the manager who puts them forth. Only a few participants dare to point this out. In the end, the manager's design ideas survive, and hours of previous design meetings are wasted. Even worse, if this pattern repeats itself, the manager comes to believe that he or she is a great designer and may take over the whole application design. Actually, this is not what the manager is paid for, and we dare to doubt that he or she does user interface design better than managing.

Usability People Can Err - Ask the Users, not the Principles

Let's face it - usability people and user interface designers can err. Often they have to make their decisions under certain assumptions and with incomplete knowledge, or they are simply expected to say something. However a decision was derived, it can be wrong, especially under such circumstances. For example, in the case of incomplete or inconsistent knowledge, interface designers often base their decisions on general rules or principles like "consistency." Users on the other hand, simply try to understand and use an application. They do not care about principles - they do not know them - and may act even contrary to them. Therefore, we urge you to ask and watch the users if you are unsure about a design decision. Conduct a user test, as quick-and-dirty as it may be, but do not rely solely on your abstract knowledge and intuitions. What matters is how well users accomplish a task, not whether certain "important" usability principles are obeyed.

The Developers Have the Power

Whatever interface designers or usability people are doing and telling the developers, they should be aware that the developers have the power, not the other way round. So, chances are good that an interface designer will advise a developer to implement a certain design, but the developer will implement something quite different. The following tips may help to improve the outcome.
Thinking Habits

This unproductive situation can happen for different reasons. Maybe, the developer simply dislikes the proposed design. More often there are technical restrictions and problems, which lead to such differences. A better communication between the developer and the interface designer would solve this, but in practice communication is not that easy. Often traditional thinking habits combine with problems which arise when technology changes. For example, when creating new Internet applications, many developers still followed the "old route" and exported ERP applications to the Web.

Designs as Feature Lists

There is another reason for differences between design proposals and the actual implementation: developers often interpret designs as a "feature list." They firmly believe that they implemented the proposed design, if they implemented most of the features - they do not have a "holistic" view of the design. Usually, user interface designers are not very pleased with the results of such an approach. They feel that the gist of the design is lost this way. If they express this feeling and complain that their design has not been implemented, this in turn confuses the developers who believe that they did what they had been told.

Coded and Standing "Solid as a Rock"

User interface designers repeatedly have the experience that once something has been coded, however flawed the design may be, it stands "solid as a rock." There are many reasons and excuses for this, such as: "It took so much time and effort to achieve the current state" or "Time pressure is high, there is no time left for changing the design. Maybe in the next release..." or more offensive "I think it's not that bad; people used it in the past and nobody complained."

One of the reasons for this disparity may be that the contact to the developers has not been close enough. If you meet developers only when they ask for help, you will be astonished what they made of your design ideas in the meantime. Aside from the reasons stated above, it is often the developers' confidence "We can solve this on our own," which leads to a "loss of contact." We already addressed the issue of talking to the wrong people. If you do not talk to the person who really implements the design, the final result might surprise you, and then it is too late for changes.

A Question of Balance

After many frustrating experiences some usability people have even come to the belief that the only way they can influence the implementation is to sit beside the developer and make sure that he or she implements the "right" design. This is, of course, not desirable, because it is just a reversal of power. What we really need is a fair relationship between user interface designers and developers in which each side accepts the other's strengths and abilities. User interface designers should not argue about SELECT statements, but developers should also acknowledge the authority of user interface designers in interface design issues - at least as a "last word" when no consensus can be found.

Other Stumbling Blocks

Sorry, Not Available Now

All design decisions end up with the selection of some "existing" interface elements, as well as their combination and arrangement on the screen, so that users can accomplish a task. We would like to draw your attention to the little, but important, word "existing." In a software company like SAP, user interface technology has at any given point in time, reached a certain state, which may be far beyond all other standards, or - more probable in the context of business applications - be a little bit behind. Usually, the company is working hard to change this, but for the developers and user interface designers this means that not all possible design options are actually available or the company's interface technology is evolving and thus constantly changing.

The latter means that developers may not be familiar with the new features or how to use them. It may also be unclear whether these features will be available in time to be implemented in the upcoming release. The features may also be very unstable in the first period of availability, something that can make development a hassle. Sad as it is, under these conditions, many great design
ideas may have to be buried when faced with reality.

Source: From Analysis to Design: Bridging the Gap
Typical Interaction Patterns

Enter Data | Inspect Data (+ Compare Data) | Manipulate Data | Search | Browse | Select Data, Functions, or Options | Initiate Action(s) | Decide | Get Help

Interaction patterns are typical ways in which users interact with a computer - irrespective of the task at hand. We suggest that there are just a few of these patterns, and that most interactions can be reduced to them. There is some overlap between the proposed patterns. For instance, initiating an action may be done by selecting a function from a menu. However, for our discussion we do not get into such details, but focus on the main purpose of an interaction.

Enter Data

One of the most common tasks for users is to enter data. Data entry can be done in a variety of ways, but doing it through the keyboard is the most common way.

We distinguish between two different situations of data entry:

- entering few data (single entry)
- entering many data (mass entry)

Entering mass data affords careful treatment, because the entry has to be done fast, efficient, and without harming users.

Inspect Data (+ Compare Data)

Inspecting data means that the users asks the system to display certain data, either for one object or for a set of objects. Depending on the task, different presentations and interaction methods may be used for inspecting data.

Browsing is a special form of data inspection with the focus on searching large sets of data.

Comparing data is a variant of data inspection where data of the same type are compared, preferably in parallel.

Manipulate Data

Manipulate data comprises a variety of ways how users can process data. Typical manipulations are:

- Move, Insert, Delete, Copy, Change

As with inspection, the manipulation methods determines how the activity is implemented and presented.
Typical Interaction Patterns

**Search and Browse**

In some cases the user cannot directly access the data that he or she wants to process or inspect. In this case there are two options:

- either the user specifies a **key** and lets the system **search** for the data, or
- the user commands the system to display a relevant set of data and **scans (browses)** the data on his or her own.

Often, both approaches are combined to make the search more effective. For example, an initial search may lead to a reduced set of data that the user scans manually. **Filtering** can be an effective means to reduce a data set to a manageable size for browsing or other types of processing.

Search is based on **remembering** certain characteristics of an object or set of objects. The better these characteristics can be used for specifying a key, the more efficient the search will be.

Browsing is based on **recognition**: Users have to recognize objects by their names or other characteristics. If hierarchical data sets are browsed, category names are also important, because users have to find out which categories are relevant.

**Select Data, Functions, or Options**

Users do **selections** for a number of different reasons:

- selecting **data** (numbers, graphical objects, text blocks etc.) for processing
- selecting **functions** for initiating actions (from menus, toolbars, pushbuttons, etc.)
- selecting **options** or attributes to influence further processing or behavior

**Note**: We reserve the pattern "select options" to the case that the system displays a set of options to the users; we do not use it in the broader sense that any user decision can be regarded as a selection of an option.

There are numerous ways to select different types of data, functions or options - you find these methods in style guides or books on user interface design. For instance, there are methods for

- single selection vs. multiple selection
- multiple selection: contiguous vs. non-contiguous selection
- combined selection and initiation: drag & drop carries out a selection and evokes a function in a single physical action

Though these selection methods are usually standardized, some of them are very sophisticated and not well known to typical users.

**Initiate Action(s)**

**Initiating action(s)** means that the user commands the computer to perform certain actions. Usually, the user enters or selects some data and then commands the computer to process them. This is how the currently prominent object-action approach works - but it might be the other way round as well.

Often, the actions take some time, and the user has to wait, until the task has been finished by the computer.
Typical Interaction Patterns

Decide

**Deciding** means that at certain stages of the processing the user has to make decisions that influence the ongoing processing. These decisions may be either

- user-initiated, or
- system-initiated

In the first and preferable case, the user is "in command". In the second case, the system prompts the user for a decision. This should occur only, either because the system lacks data, or because an emergency situation has arisen.

**Note:** We reserve the pattern "Decide" to the case where the user has explicitly to determine how the flow of processing will proceed; we do not use it in the broader sense that any user action can be regarded as a decision.

Get Help

This interaction is important from the user's point of view, though it is usually not directly integrated into the processing of the task.

As we noted in "Decide", the system may lack data and prompt the user. Here, it is the other way round: The user lacks information and asks the system for assistance.

As this behavior is usually separated from the normal course of processing (the system prompt is not!), there are additional mechanisms like on-line help that serve the purpose of providing help. Note however, that it is better to make an application self-explanatory than to have to rely on additional help mechanisms.

**Source:** *From Analysis to Design: Bridging the Gap*
Elementary Tasks

Here we list elementary tasks, focusing on those which are relevant for business applications. Real-world tasks are composed of one or more of these building blocks.

Selection

Users have to select objects and functions in order to accomplish their tasks. In the currently dominant object-action approach users first select the object(s) to process and then the action. However, depending on the task this may not always be the most efficient way to do the task.

Object selection

Object selection may result in a rather complex process if objects have to be browsed or searched, are unknown, belong to large data sets etc. For object selection we distinguish between

- **object specification** (new objects, well known objects): usually done by entering a name or clicking at an object,
- **object search** (incomplete object specification) (existing objects, partially known objects): usually done by entering a search string = (partial) name for the object,
- **object browsing** (visual search through sets of existing data): usually done by scanning larger structures of text or graphical objects that are only partially visible.

**Specification** may be combined with search or browsing, if new objects are based on (copied from) existing ones. **Search** may result in a hit list which then serves as basis for object selection. If the hit list is longer, it may be browsed for candidate objects. **While browsing** is primarily based on recognition, search and specification are primarily based on recall which is usually harder for people.

Filtering

Whenever there are larger lists of object, either created through a search or presented by the system, **filtering** allows users to restrict the object set to a manageable size and useful subset. Filtering can already be done by the system for **preselection** of a useful object set (the criterion may be set by the users), or **manually by the user**; often it makes sense to combine both approaches.

**Note**: The "shuffler" promoted by Alan Cooper is a "ready-to-use" interface pattern for filtering. However the details of the filter depend on the context of the task at hand.

Selection Modes

The most common task in object selection is **single** selection, e.g. selecting just one object for further processing. Other tasks require the selection of two or more objects (**multiple selection**). For multiple-selection we also may have to distinguish between continuous and discontinuous selection, the latter introducing more complexity and handling problems than the first one.

Also **sets of objects** may be single-selected as a group or partial group (depending on attributes/filters).
Function selection

Function selection usually is done after one or more objects have been selected (object-action approach). Functions are typically selected through some sort of function menu, e.g.

- functions in pulldown menus
- functions in toolbars or palettes
- functions on pushbuttons located closely to the affected object(s)

Often menu functions can also be initiated by a key code instead.

Another approach to function selection is to use a direct-manipulation mechanism like drag&drop: Here objects are selected and dragged onto another screen object in order to perform a function. Yet another approach is to issue textual commands like in command-driven interfaces.

Usually just one function is selected for processing. However, users may also select a sequence of operations for more complex tasks. Such sequences in turn may be compiled into one operation (a macro).

Function selection is strongly impacted by the presentation of the functionality: If users cannot find a function, they cannot select and use it!

Elementary Processing - Single and Multiple Processing of Objects

In business transactions the basic processing comprises data entry, display and editing (master data, transaction data). Two main processing modes have to be distinguished:

- **single** processing (create, display, edit),
- **multiple** processing (display, edit).

There are also modes possible that lie between these two extremes, namely

- parallel processing of few (two or more) objects.

Single processing is usually handled in a **template view** (using **fields**), while multiple processing is handled in **tables**.

For multiple processing, a **detail view** can be provided (usually in a template view) where users can edit more data than the table allows to display. This detail view may replace the overview table or complement it. Navigation between items in detail view may be solely done via the overview table (usually as random access) or also in the detail view (in this case usually in a sequential fashion).

For processing a few objects in parallel, these objects may be presented either sequentially (e.g. in a fixed screen area), in parallel as templates (e.g. in screen tiles) or in parallel as a small table with more fields than are usually presented in a table view.

User Perspective

From the **users'** point of view these tasks involve **routine** or even highly **automatic** processing, but usually no problem solving (apart from finding out which values to enter in a field...).
Elementary Tasks

However, also **untrained** and **casual** users may have to do these tasks. In this case screens have to be intuitive and to provide enough context and procedural information (external knowledge) so that these users also can manage such tasks.

**Advanced Processing - Handling Relations Between Objects**

Handling relations between objects often is a complex task and may require more or less problem solving. Also the presentation requirements are more demanding, because usually several objects have to be displayed in parallel. Also the ways objects can be put in relation to each other influences how data have to be presented.

Tasks that require little problem solving and can be done by untrained and casual users again have to provide enough external knowledge to enable these users to accomplish the task.

Presentation and interaction may be very varied and flexible for these kinds of processing. Therefore, we provide just a few examples:

- **Inserting** objects (from a separate set) into an object list or hierarchy
  Example: A carrier has to assign and reassign deliveries to shipments.
- **comparing** two or more objects with respect to one or more attributes
  Example: A customer has to compare products from different vendors with respects to features and prices.
- **copying** objects as a template for new or existing objects,
- **combining** objects (e.g. steps) into new objects (e.g. procedures)
  Example: An engineer has to assemble parts lists for products.
- **presenting** objects in diagrams or other representations (lists, graphics, ...).

**Combined Processing**

These types of processing may be combined with the simple processing tasks mentioned above. For example

- a cost center may be created, inserted into a hierarchy, and then provided with data.

As this example shows, the assignment to a hierarchy might also be done later. So, an important design issue might be, which of the subtasks have to be offered in parallel, or whether a separation on steps might be more transparent and efficient.

In such cases, space requirements are even more demanding.

**User Perspective**

From the users’ perspective these tasks may require more assistance or even guidance, especially if users are untrained or use the software only casually. Proficient users may want to do without system assistance, though there may be cases where even these users appreciate system assistance.

Important issues for demanding tasks are **directness** of interaction, **error robustness**, **flexibility**, and - of course - **efficiency** (time, steps).
From Interactions to Applications - The R/3 Way

Definitions | Table Overview

Below we provide a bottom-up view that is somewhat different from the view in Part III: Defining the Design. It corresponds more closely to the view that prevailed in the R/3 system and which lead to a classification of screens and applications based on primary interactions. New to this older view is the addition of elementary patterns like the before-mentioned interaction patterns. We find that at least for some interactions there is a connection that goes from interaction patterns to screens and further to applications. That is, screens and applications had been classified on the basis of their primary interaction mode.

Definitions

An interaction pattern is a prototypical interaction that is found in many tasks and that forms a basic building block for the communication between the user and the computer. It is still abstract in the sense that it does not describe the detailed behavior of the user.

Prototypical screens are screens that are devoted to certain prototypical tasks. As you see in the table below, often these screens directly relate to interaction patterns.

Application patterns are applications or partial applications built from those building blocks. They may consist of one or more prototypical screens that are connected to perform a prototypical task. An application that consists of one or more of these pattern may be termed a "typical" or "prototypical" application (or an application type).

Table Overview

The table below displays the relation between interaction patterns, screen types and application patterns for "traditional" R/3 applications.

<table>
<thead>
<tr>
<th>Interaction Patterns</th>
<th>Screen Types</th>
<th>Application Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter Data (few, many)</td>
<td>Entry Screen (single, collective)</td>
<td>Single Data Entry (fields), Collective Data Entry (table)</td>
</tr>
<tr>
<td>Select Data</td>
<td>Initial Screen, Selection Screen</td>
<td>Any Application Pattern</td>
</tr>
<tr>
<td>Select Functions</td>
<td>Data Screen and Variants, any Screen Type</td>
<td>Any Application Pattern</td>
</tr>
</tbody>
</table>
From Interactions to Applications - The R/3 Way

<table>
<thead>
<tr>
<th>Task</th>
<th>Screen Type Options</th>
<th>Application Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Options</td>
<td>Data Screen and Variants, any Screen Type</td>
<td>Any Application Pattern</td>
</tr>
<tr>
<td>Initiate Actions</td>
<td>Any Screen Type</td>
<td>Any Application Pattern</td>
</tr>
<tr>
<td>Decide</td>
<td>Any Screen Type, Prompt Dialog</td>
<td>Any Application Pattern</td>
</tr>
<tr>
<td>Inspect Data, Manipulate Data</td>
<td>Data Screen, Additional Data Screen, Detail Screen, Information Dialog</td>
<td>Display Data, Edit Data</td>
</tr>
<tr>
<td>Search</td>
<td>Selection Screen + Overview Screen; if needed Detail Screen</td>
<td>Display Data, Edit Data</td>
</tr>
<tr>
<td>Browse</td>
<td>Overview Screen</td>
<td>Data Browsing, Display Data, Edit Data</td>
</tr>
<tr>
<td>Request Help</td>
<td>Help Screen, Information Dialog</td>
<td>Any Application Pattern</td>
</tr>
</tbody>
</table>

For details see:

- interaction patterns
- screen types
- application patterns

There is also a relation between application patterns and basic **application structures** as well as their **presentations**. However, this is not a direct relation, as a structure may be presented in different ways, depending on the nature of the actual task at hand. However, these relations can help you to select the right presentation for typical tasks and applications.

**Source:** From Analysis to Design: Bridging the Gap
Screen Types

Initial screen | Entry Screen (Single Entry, Collective Entry) | Data Screen | Additional Data Screen | Detail Screen | Overview Screen | Selection Screen | Prompt Dialog | Information Dialog

The following typical screens are mainly based on interaction patterns or on more complex but generic tasks. Most of these screen types correspond to the screen types that were defined for the R/3 system.

Initial Screen

This screen usually appears at the beginning of the processing or at the beginning of the processing of certain data.

If this screen appears at the very beginning of a computer session, it usually prompts the user for identification. This screen may be the main screen, but might also be implemented as a prompt dialog. Usually such an initial screen is only loosely connected to the other screens of an application - it is just the starting point.

If the initial screen appears at the beginning of a certain task - like in the R/3 system -, the user usually has to specify the object or object class to be processed. In some R/3 applications the user also specifies the type of processing and/or other relevant data.

Initial screens are usually the starting point and sometimes also the ending point for an application.

Basic Task: Entering data for user identification, object specification, and/or specification of processing.

Entry Screen (Single Entry, Collective Entry)

An entry screen serves the purpose of data entry. As with interaction patterns we distinguish screens for entering few data from screens for mass data entry (collective data entry). The main difference between both screen types is that

- screens for single entry are used for entering data for one object (which may be quite a number of data -> master data); speed is not the most important factor in this case
- screens for collective entry are used for entering data for many objects; usually, there are several data entered for each object, but not as many as with single entry; speed is an important factor for these screens.

Usually screens for single entry are laid out with entry fields, while screens for collective entry use tables.

Basic Task: Entering data for single objects or for many objects.

Data Screen

Data screens are used for data display. There are some important variants of these screens that are covered below. Data
screens are related to the interaction patterns of inspecting and browsing data.

In the R/3 system there is a dualism between data screens and entry screens. Here screens for data display and editing have the same basic layout and can be toggled between display mode and edit mode.

In general, data screens may be used for very different types of data: textual or numerical data, longer texts, graphics, and a mixture of all of them.

**Basic Task**: Inspecting single objects.

### Additional Data Screen

This is a variant of the data screen. It is a separate screen that displays additional data which complement the data in the main screen.

In the R/3 system additional data screens usually replace the current screen, that is, data screen and additional data screen are displayed alternatively in the same primary window. Additional data may also be displayed in modal dialog windows. In this case, users have to close the dialog window in order to continue processing in the main window.

However, it is far more useful, to display additional data in parallel windows, either in parallel primary windows or in amodal dialog windows. This way the additional data can be viewed together with the basic data, and the main processing can continue without disturbance. As one caveat, additional windows increase the complexity of the user interface. This problem has to be balanced with the gain in transparency through the parallel display of data and additional data.

**Basic Task**: Inspecting additional data of single objects.

### Detail Screen

The detail screen is a frequently used variant of the data screen. It displays detail data for a selected object.

In the R/3 systems, there are the same limitations for detail screens as for additional data screens - usually they replace the main screen, where the object has been selected for which detail data are displayed. Detail data may also be displayed in modal dialog windows.

As with additional data screens it is often more useful to display detail data in parallel windows.

**Basic Task**: Inspecting detail data for single objects.

### Overview Screen

The overview screen is another common variant of the data screen. It displays overview data, for instance an overview of the components of an object.

In the R/3 systems, there are the same limitations for overview screens as for additional data screens - usually they replace the main screen.
Screen Types

As with additional data or detail screens it is often more useful to display overview data in parallel windows.

**Basic Task:** Browsing objects.

**Selection Screen**

The Selection screens is a variant of the entry screen. It is devoted to the selection of one or more objects for processing or inspection. It may include very sophisticated means for data selection.

Selection screens are useful, if large amounts of data are to be processed; they serve as a filter to the data and help to reduce the amount of data that is displayed. They also help to find very specific data.

**Basic Task:** Selecting single objects or sets of objects, searching for objects.

**Prompt Dialog**

A prompt dialog is a system-initiated dialogue, usually a modal dialog window, that prompts the user for entering some needed data, or for confirming an action or decision. Depending on the purpose of the prompt, users have to enter more or less data, or just to hit a button to confirm or abort the dialog.

**Basic Task:** Entering special data.

**Information Dialog**

Information dialogs are system-initiated windows that inform users, for instance about the success of an action, about the ongoing processing, about the system state etc. Usually, users just confirm such dialogs and do not enter any data.

**Basic Task:** Displaying information (data).

**Source:** From Analysis to Design: Bridging the Gap
Application Patterns

Single Data Entry | Collective Data Entry | Data Edit | Data Display | Data Browsing

**Application patterns** are applications or partial applications that are built from one or more prototypical screens and/or interaction patterns. Application patterns act as a unit that performs a prototypical task. An application that consists of one or more of these patterns can be called a "typical" or "prototypical" application (or an application type).

**Single Data Entry**

This pattern describes a simple application for entering data for **one or more single objects** on one or more screens at a time. That is, either

- one object may use one or more screens, depending on the number of data to be entered
- further objects may use additional screens (each object uses one or more screens)

A typical pattern for such an application comprises

- an initial screen for specifying the object
- one or more entry screens
- if more than one object is to be processed:
  - a mechanism for specifying additional objects
  - entry screens for these objects (usually the entry screens are reused)
- optionally an exit screen (may be the initial screen)

Possible extensions for this pattern are

- detail screens for displaying details of objects
- overview screens for providing overviews over objects, i.e. their components, or over the objects being processed

**Structure**: This application pattern can be implemented as a sequence with higher-level entry and exit screens. This sequence may have side paths for detail or overview screens.

**Collective Data Entry**

This pattern describes a simple application for **entering mass data** on one screen.

A typical pattern for such an application comprises

- an initial screen for specifying the object
- one entry screen for mass entry (with table, list, etc.)
- optionally an exit screen (may be the initial screen)

Possible extensions for this pattern are
Application Patterns

- detail screens for displaying details of selected objects
- overview screens for providing overviews over objects, i.e. their components, or over the objects being processed

**Structure**: This application is very simple and consists basically of one screen. This screen may have a higher-level entry and exit point as well as side paths for detail and overview screens (actually the entry screen is an overview screen).

**Data Edit**

**Data edit** often is handled in the same way in the SAP system as data display which is described below. The main technical difference is that fields are open for editing and thus unprotected.

**Data Display**

**Data display** is a very varied task that depends on the structure and number of objects to be inspected. We distinguish between displaying

- single objects in more or less full detail
- a set of objects in more or less full detail
- a set of objects with little detail

When displaying more than one object, we also have to take the structure of the set of objects into account. The objects may be structured as an unordered list (set), as an ordered list, as a hierarchy, or even as a network.

**Displaying Single Objects**

In this case, one or more screens with entry fields form the core of the application.

**Structure**: This application is very simple and consists basically of one screen or a sequence of a few screens. This screen (or these screens) may have a higher-level entry and exit point.

**Displaying a Number of Single Objects**

In this case, the one or several screens for one object are "reused" through the object set. There are mechanisms needed for navigating between the objects.

**Structure**: This application consists of a sequence of screens which may have subsequences, if more than one screen is needed for one object. This sequence may have a higher-level entry and exit point.

**Displaying Many Objects (Partial Set, Whole Set, ...)**

If many objects are to be displayed, solutions similar to mass entry (usually based on tables or lists) or data browsing (usually based on hierarchies) may be used.
**Data Browsing**

Data browsing is similar to data display. However, with browsing the focus is on search, that is on selecting one or more objects from a large set of objects. Therefore, the most important issues are navigation within the data set and identification of objects (and categories, if the data set forms a hierarchy). Once the objects are identified (found), the browsing task has been done, and the objects may be further processed, i.e. inspected, edited, copied, deleted, etc. Thereafter, the browsing may continue.

A typical pattern for a browsing application comprises

- an initial screen for specifying the data set to be browsed
- one or more screens for browsing though the data
- optionally one or more screens for displaying and/or editing the found objects
- optionally an exit screen (may be the initial screen)

Possible extensions for this patterns are

- detail screens for displaying details of found objects
- overview screens for providing overviews over objects, i.e. their components, or over the objects being processed

**Structure:** There are many variants possible for browsing applications. A very primitive version might consist of one screen that displays the list or hierarchy of objects to be browsed. This screen might be replaced by a display/edit or detail screen for found objects. This main screen may have a higher-level entry and exit point.

There are many extensions possible for this simple application:

- The display/edit may be done in a separate screen.
- A split-screen design may be utilized, displaying the set of objects on the left and the selected object on the right. The implementation of this split-screen depends on the technology used; on the web, for example, a frame design would serve this purpose.
- If the data set is hierarchical, one of the many presentation forms for hierarchies can be used. The actual choice depends on the task requirements.

**Source:** From Analysis to Design: Bridging the Gap
**Application Structures**

Simple One-Screen Application | Sequence | Hierarchy, Tree | Network | Complex One-Screen Application: Compound Screens

In the following we list typical structures for applications. A special case is the compound screen: This is a one-screen design with a more complex inner structure that is comparable to a multiple-screen design - and also an alternative to it. Note also, that any of the multiple-screen structures can be extended to include compound screens.

**Simple One-Screen Application**

**Definition**

One-screen applications present all their functionality on one screen, dialog window, or web page. This design is suited to small, well-defined tasks only.

**Examples**

- Calculator
- POS application (point-of-sale) with a touchscreen

**Implementation, Presentation**

As one-screen applications present all their functionality on one screen, a careful design of the screen is very important.

One-screen applications can be made more versatile by exchanging the contents of certain screen areas during the course of processing. Use this feature with care, because it may also make using a one-screen application more complex and thus easily lets a simple one-screen application slip into becoming a complex one-screen application that has other uses and users.

**Sequence**

**Definition**

Sequences consist of a chain of screens, dialog boxes, or pages that have to be processed either in a fixed or arbitrary order.

There may be dependencies between screens that dynamically change the sequence, e.g. screens may be skipped or added. An arbitrary sequence is actually "a net in disguise" (network structure), where each screen is connected with each other.

Often a sequence is entered from a higher-level screen. The processing may return to this screen or not.

**Examples**

- Wizard (strict sequence of dialog windows),
- data entry applications that are distributed over a number of screens (usually strict sequence)

**Implementation, Presentation**
Screen Chain

A sequence of screens, that are processed in strict order. The screens may be primary screens, web pages etc.

The screen transitions may be provided by specialized functions or links and/or by simple navigation functions or links (forward, backward).

Side paths may be possible, but should return to the source screen.

**Typical uses:** Data entry or display for complex objects.

"Long" Screen

A scrollable screen or web page that comprises all processing steps or data for one object.

**Note:** This implementation does not enforce a strict order, however it suggests such an order through its linear structure.

**Typical uses:** Data entry for complex objects, display of complex objects (or long text).

Wizard

A sequence of views with "standardized" forward and backward navigation.

Wizards are often implemented as dialog windows (but need not be). Ideally wizards provide previews for the choices the users make.

Side paths are usually not possible.

**Typical uses:** Procedures like installing new hardware, changing system settings etc. where users need assistance from the system and/or where user errors are critical.

Hierarchy, Tree

**Definition**

Many applications or information systems consist of a hierarchy of screens.

In some cases, the whole hierarchy has to be processed, either in a fixed or arbitrary order. In other cases only parts of the hierarchy have to be processed.

A strict hierarchy demand that users navigate within the hierarchy only. Users cannot navigate across the hierarchy, but have to backtrack, when they want to enter another branch of the application tree.

**Example**

A typical R/3 application usually has a hierarchical structure consisting of an entry screen, a number of data screens, which may gain lead to detail screens etc.
**Application Structures**

**Special Case: Hierarchical Sequence**

This structure is a tree with simple sequences at the nodes.

**Example:** A set of tasks that are organized as a hierarchy, while each task is processed as a simple sequence, an on-line documentation that is hierarchically organized into chapters and subchapters, while each chapter is a sequence of pages (screens).

**Special Case: Sequential Hierarchy**

This is just the opposite of the previous structure. Here, the basic components are arranged in a sequence, while each component itself may have a hierarchical (or a simpler) structure.

**Example:** An application that uses a list as basic organizational structure. The list elements are structured in some way that forms a structure below the basic list (or sequential) level. For example, lower levels may present detail screens, partial objects etc.

**Implementation, Presentation**

From the users' point of view an application with a tree structure looks similar to a screen chain, because

- the screen changes are provided through function calls,
- the hierarchy is not visualized.

Usually there are no standardized navigation functions (forward, backward), as the navigation in the tree is more complex than in a sequence. As the hierarchy is implicit only and hidden from the user, there is also no indication of where a user currently is located within the hierarchy (with respect to the whole hierarchy).

A tree structure may be extended by providing cross-links between branches which actually transforms the tree into a more or less restricted network structure.

**Typical uses:** Data entry or display for complex objects.

A strict hierarchy may also be implemented as a **stack** metastructure.

**Network**

**Definition**

A network structure is an application structure in which screens or pages can be arbitrarily linked.

A simple network consists of a number of screens where users can navigate from one screen to any other screen.

Often, there are restrictions that prevent users from going every possible path. A hierarchy with cross-links is a typical example for a network structure: Here users can jump from one branch in the tree to another without having to backtrack.

**Examples**
Application Structures

- Simple web sites where users can access any other screen from each screen,
- property sheets for setting attributes or options (arbitrary sequence, an arbitrary number of screens has to be processed),
- hierarchical R/3 applications with cross-links,
- complex hypertexts.

Implementation, Presentation

If you implement a network structure, you have to consider:

- the size of the network, e.g. the number of nodes (screens or pages)
- the "connectivity", e.g. can all screens be accessed from each node or are there restrictions to the navigation?

Fully Connected Simple Network

A simple, fully connected net can be implemented as

- screens with pushbuttons, menus or links that provide access to all relevant screens,
- tabstrips with tabs that provide access to all relevant views; this is the preferred presentation for property sheets.

Such structures are transparent as long as the number of navigational options is limited to a small number.

Typical uses: Small web sites or web applications, editing or display of complex objects with several view options. There should be no dependencies between screens or views that restrict the navigation.

Large Partially Connected Network

Large networks can provide severe problems for users, if there is no guidance and no indication of their location within the net.

Usually large nets are implemented like hierarchies with additional cross-links between branches, that is, these nets do not have "full" connectivity.

If large nets are based on data structures, an index or stack design is more appropriate.

Typical uses: Data entry or display for complex objects.

Complex One-Screen Application: Compound Screens

Definition

One-screen applications are not restricted to simple applications. More complex applications may as well present all their functionality on one screen or web page. In this case, however, the screen is divided into separate areas that serve certain purposes. These areas correspond to screens in a multiple-screen design and are usually connected with each other in some sort of dependency. The screen usually not only has a vertical structure, but also horizontal divisions.

Note, that a compound screen usually is based on a metastructure like an index or stack as a organizational structure. See Metastructures for Applications for more information on this aspect!

Examples
Application Structures

- Complex R/3 application for managing master data
- Self-service application for address data

Implementation, Presentation

Compound screens can be implemented using different techniques depending on the platform used (e.g. subscreens in R/3, frames, iframes, or DIVs on web platforms).

For the presentation we propose the following:

- Name the areas according to their purpose and display titles for them
- Separate areas visually so that the screen looks "organized"
- Arrange areas logically according to task flow and dependencies
  - Flow: Top to bottom, and/or left to right
  - Dependency: Top to bottom, and/or left to right

Do not reverse the flow of control in order to reuse areas!

A typical implementation for areas is the "tile" paradigm, where areas have fixed rectangular locations. Areas may be expandable and collapsible to use screen space more efficiently. However, these "instabilities" make screens more complex. Screen areas might also be used for different purposes; here the same pros and cons apply.

Pros and Cons

Screen areas allow parallel presentation and processing of data. This avoids screen changes and navigation, and thus makes working with compound windows more efficient. In addition, working memory load is lower, because users need not remember data between screens.

Efficiency decreases, however, if there are too many areas on a screen or page and if the single areas are too small for efficient work. This happens, for example, when the areas are so small that users are forced to scroll areas, or if users have to excessively scroll tables within an area, because only a few lines can be displayed at once.

If designed well, compound screens are well suited to experts and frequent users. But their visual and functional complexity as well as the often unclear flow of control and the dependencies among screens, make them less appropriate for beginners and casual users.

There are, however, exceptions to this rule! If applications are easily understood by beginners and casual users, like for example address data maintenance, they can also utilize the compound screen design in order to increase efficiency and minimize navigation.

Source: From Analysis to Design: Bridging the Gap
Compound Screen Layout Examples

Single Screen | Vertical Division | Horizontal Division | T-Shape | Complex T-Shape

Below we present typical screen layouts for compound screens. The areas in compound screens may have very different uses. However, in R/3 transactions and in many Easy Web Transactions their usage is fairly standardized; therefore we indicate typical functions for these areas in our diagrams using the following colors:

- Work Area, Overview Area
- Header Area
- Navigation Area
- Detail Area
- Message Area
- Application Header (generic)
- Multi-purpose Area (generic), e.g. Help

In the diagrams we display an additional application header that might carry menus, screen title and toolbars.

Single Screen

Single screen applications put all functionality relatively unstructured on one-screen (at least there are no major subdivision).

Vertical Division
The work area may be subdivided vertically, for example to separate a header area from a work or information area:

**Horizontal Division**

Alternatively, the screen may be divided horizontally, for instance to display a navigation area:

**T-Shape**

These following screens are examples for more complex applications. A T-shape layout combines vertical and horizontal divisions showing, for example, a header area on top and a navigation area beside the work area.

**Complex T-Shape**

Of course, screens subdivision may be arbitrarily complex. However, you should refrain from too complex subdivisions, because the more areas a screen has, the harder it becomes for users to understand their purposes and relations. In addition, the areas become too small to be used efficiently. Here we present two more examples based on the previously shown subdivisions:
Compound Screen Layout Examples

And with a header area for the whole screen:

Source: From Analysis to Design: Bridging the Gap
Index | Stack | Queue

Metastructures are compound structures with screens divided into several areas that serve different purposes. Usually one of these areas contains an organizing element, that aids in accessing screen areas or screens - in the cases below this is an index, a stack, or a queue mechanism. Often there are no (or only few) screen changes, but only certain areas change their contents according to the access actions.

Index

Definition

An index structure is a metastructure, characterized by a static index, that is based on the application structure or a central data structure to which the index provides access. Thus the index can best be viewed as an "access" structure or device. An index is, however, not restricted to being static; it may also change over time, if new elements are added to the index or old ones are deleted, moved or changed.

The index is used for navigation within the application or data structure, that is, it provides access to its components. Usually the index is permanently available, but it may also - for example on user request - be hidden from view.

In the simplest cases the index may be a list or a tree. But there are alternatives:

- graphical index: map (2-dim), collection of objects (e.g. icons, 1-dim, 2-dim)
- mixed structure (text and graphics or diagrams): time line (1-dim), road map (1-dim, 2-dim), chart (2-dim)
- text-based structure: calendar (2-dim), timetable (2-dim)

While the index as a whole is static, its display may be dynamic. For instance, if the index is a tree, parts of it may be collapsed or expanded.

The objects that form the nodes of the index may have a more or less complex internal structure themselves.

Usually an index implies a random access to its elements!

Examples

- A simple index structure may consist of a number of permanently available buttons that give access to all screens, pages or views of an application.
- The table of contents of an online-handbook may serve as index for browsing the documentation.
- An R/3 application may have a list or a hierarchy of objects as "organizer" and navigational device.
- A time management system may have a calendar as basic access device to time-related data.
- A sales application may have a map as navigation structure for location-related data.

Implementation, Presentation

Applications with an index structure preferably use some sort of compound window, e.g. a fixed window structure with
Metastructures for Applications

subwindows (see the on-line handbook example). There are many possibilities to implement this general layout.

Few navigational options

If there are only few navigational options, a simple list, graphics or button bar may serve to present the options. Therefore it is often not necessary to reserve a special area or subwindow for the index.

- **Tabstrip**: Tabs display the navigational options (views); the working area of the tabstrip displays the data.
- **Buttonbar**: A buttonbar or toolbar displays the navigational options; the work area of the screen displays the data.
- **Floating navigation palette**: A floating palette window may also be used to display the navigational options. This way the data window is not disturbed by the navigation functions.
- **Simple web applications**: Buttons or lists of links display the permanently available pages. The navigational options are usually displayed redundantly, e.g. at the top, at the bottom, graphical and textual.

**Typical uses**: Small web sites, applications for setting options

Many navigational options

If there are many navigational options, a list or tree is used to display the index which appears in a separate window or subwindow. The index may have to be scrolled, if it is too large to fit the subwindow.

- **Split screen design**: One screen area or subwindow is reserved for the index or index graphics, other screen areas display the data, buttons, etc.
- **Separate index window**: The index may also appear in a separate window that is displayed in parallel with the data window.
- **Complex web applications or information sites**: Frames are used for the index and the data; you can introduce additional frames for header information, navigation functions, etc.

**Typical uses**: Applications for inspecting and editing large data structures, documentation viewers, information-based web sites.

Stack

**Definition**

A stack is a dynamic metastructure with restricted navigational options. Instead of providing a representation of the whole structure, as an index does, a stack displays only the path to the current screen or page. The path provides users with the context of the current page and can also be used as an efficient means for mowing backwards or up the hierarchy. Forward navigation has to be provided through functions analogue to those in sequences or trees.

Because users navigate through an application and thus change the path, a stack is dynamic. The path comprises the sequence of screens that lead to the current screen; it is not a list of the actual movements of the user, as is the case in web browsers.

**Examples**

- An application for goods issue in a factory could, for instance, use a stack for navigating through the part hierarchy of a car or electronic or mechanical device.
- A hotel application using a touchscreen could display the drinks and meals that a barkeeper orders from the goods issue outlet; drinks and meals are organized hierarchically, and the barkeeper wants to steps quickly and with simple finger clicks through these hierarchies.
- An information website with deep hierarchical structure might display a stack so that users can back up to other levels without having to step through the full path.
Metastructures for Applications

Implementation, Presentation

A stack usually has a toolbar or button bar that holds the buttons or links for the navigational options, that is, the screens that lie on the path between the entry screen and the current screen. The screen itself holds the data. With this implementation, there is no big difference in layout between a sequence or tree and a stack.

Stacks can also be visualized similar to tabstrips with the tabs being added or deleted dynamically. The simplest way, however, to display a stack is showing a text-based path info. The path items can act as links so that users have access to any location on the path.

Typical uses: Data entry or display for complex objects. A stack is useful, if there is no cross navigation, that is, if the application structure is a strict sequence or tree.

Queue

Definition

A queue is a dynamic metastructure of fixed size. While a stack follows the “first in - last out” principle, a queue follows the “first in - first out” principle: A queue is a list of items where new items are added to the end, and front items drop out as soon as a certain queue size has been reached. Usually a queue will contain only 5-10 items, all of which are directly accessible. Users may also move forward or backward between the queue items.

There is a variant of the queue where the foremost item may not drop out as soon as a new entry comes in. In this case the items are ordered by some ranking principle, and the item with least priority drops out. This item may reappear - depending on the context - as soon as there is a slot in the queue available again.

Examples

- A monitoring application may use a queue for displaying the most urgent problems. Here the queue is ordered by priority. Items drop out, after a problem has been fixed or if more urgent problems appear. These items will reappear in the queue as soon as there is a vacant slot in the queue.
- A business application may display the most important customers, regions, products, sales volumes etc. Again, items are ordered by ranking, e.g. sales volume, and items with low rank drop out as soon as items with higher ranks come in.

Implementation, Presentation

As a queue is of limited size, it is fairly easy to implement. The access to the items can be presented as a list of object names, a list of links, buttons, or graphics, as a map, or through any other direct access mechanism. Often this presentation can be combined with some sort of status information. The items themselves can be displayed in a reserved area similar to an index layout or a stack design. An item wise forward and backward navigation can also be provided in this screen area.

Typical uses: Monitoring applications.

Source: From Analysis to Design: Bridging the Gap