



Dokeos User Interface Guidelines 1.0



Copyright © 2006 Dokeos
Rue des Palais 44 Paleizenstraat, B-1030 Brussels, Belgium
info@dokeos.com
Tel. +32 (2) 211 34 56

Permission is granted to copy, distribute and/or modify this document under the terms of the *GNU Free Documentation License*, Version 1.2. You may obtain a copy of the *GNU Free Documentation License* from the Free Software Foundation by visiting their Web site [<http://www.fsf.org>]





Table of Contents

Introduction	5
Interface User Guidelines	6
1. Involve Users in the Design Process	6
2. Speak the users' verbal and visual language	7
3. Build on users' Mental Model of the tasks	8
4. Keep the User Informed	9
5. Provide appropriate feedback	10
6. Put the User in Control	10
7. Make Actions predictable and reversible	11
8. Reduce latency	12
9. Avoid feature bloat	13
10. Manage Complexity	13
11. Make Your Application Consistent	14
12. Provide an aesthetic Integrity	15
Subtractive design	16
Visual hierarchy.....	16
Extra space.....	16
Icons	16
Fonts	16
Colors	16



13. Allow users to customize	17
14. Make "always at the same place" illusion	17
15. Support Direct Manipulation of objects	18
16. Shorten labels with the key word(s) first	18
17. Reduce the distance of successively used elements	19
18. Prevent errors	19
19. Provide help and documentation.....	20
20. Be aware of cultural differences	21
21. Be aware of linguistic differences.....	22
22. Don't exclude people with special needs	23
Visual Disabilities.....	23
Other disabilities.....	24
23. Be conformant to public norms.....	23
Références	25



Introduction

The present document is an effort to provide a framework of important guidelines for Dokeos interface design to help developers and designers make more effective contributions to Dokeos projects.

The goal of user interface design guidelines is to have the interface positively support users' endeavours and to make the interface efficient, satisfying and transparent to the task the user is trying to accomplish. Respecting the guidelines can significantly improve the quality of Dokeos user interface in the following ways:

- ❑ Users will learn faster how to use Dokeos if the interface looks and behaves like applications they're already familiar with
- ❑ Users can accomplish their tasks quickly, because well-designed applications don't get in the user's way.
- ❑ Dokeos applications will answer more effectively to users' real needs
- ❑ Dokeos will seem less complex and simpler to use
- ❑ Users will make less errors and consequently will find Dokeos more stable, satisfying and fun to use
- ❑ Dokeos will be more compliant to work with old, current and future technologies
- ❑ Users with special needs will find Dokeos more accessible
- ❑ Dokeos will be easier to document, because an intuitive interface and standard behaviours don't require as much explanation
- ❑ Dokeos will be easier to localize and internationalise

User interface principals are the competitive differentiator in a now crowded marketplace. Those companies that position the user satisfaction as a strategic part of their business will survive and prosper. Effective interfaces are visually apparent and forgiving, instilling in their users a sense of control. Users quickly see the breadth of their options, grasp how to achieve their goals, and do their work.

The following design principles are based on web usability and accessibility principals and on insights from software engineering, linguistics and psychology. These guidelines are prepared to assist you in developing products that provide Dokeos users with a consistent visual and behavioural experience across applications.



Interface User Guidelines

1. Involve Users in the Design Process



The best way to make sure your product meets the needs of your target audience is to expose your designs to the examination of your users. Doing this during every phase of the design process can help reveal which features of your product work well and which need improvement.

There are two major sources of information about people: what people say and what you can observe. Don't trust either source exclusively. Observing and talking produce different cross-sections of information which will illuminate one another.

Exposing a prototype of the feature you are developing to some users may help you discover problems that you did not anticipate during your initial design phase. Clearly identifying the needs of your users helps you create products that deliver effective solutions and are typically easier for them to learn and use.

Recognize that, as an application developer or interface designer, you have a greater wealth of knowledge and a more intricate understanding of your application than your customers are likely to have. Although you should use that knowledge to choose the best default settings or decide the best presentation of information, remember that you are not designing the program for yourself. It is not *your* needs or *your* usage patterns that you are designing for, but those of your (potential) customers.

Try not to have any preconceived ideas other than to provide users with the tools they need. Try to develop an awareness of: the person's goals (long and short term), what they pay attention to and what they ignore, major sources of irritation, their physical environment, who they interact with and how, what actually dominates their time, and how existing products shape their work.



2. Speak the users' verbal and visual language



The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms and concepts. Follow real-world conventions, making information appear in a natural and logical order.

Real-world representations and natural interactions give the interface a familiar look and feel and can make it more intuitive to learn and use.

You can often take advantage of your users' knowledge of the real world by using metaphor— that is, a familiar concept from the outside world— to represent elements within your application. For example:

- ❑ An image of a file folder suggests a container into which documents can be placed
- ❑ An object displayed as a trash can communicates to users that it is a place for discarding things

When using metaphors, however, it is important to neither take the metaphor too literally, nor to extend the metaphor beyond its reasonable use. For example, the capacity of a file folder should not be limited to the capacity of a physical file folder, which presumably could contain only a few documents before becoming unwieldy. On the other hand, a waste basket should not be used for anything other than holding discarded files.

There are several factors to consider when using a metaphor:

Once a metaphor is chosen, it should be spread widely throughout the interface, rather than used once at a specific point. Even better would be to use the same metaphor spread over several applications.

Metaphor isn't always necessary. In many cases the natural function of the software itself is easier to comprehend than any real-world analogy of it. Don't strain a metaphor in adapting it to the program's real function. Nor should you strain the meaning of a particular program feature in order to adapt it to a metaphor.

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



3. Build on users' Mental Model of the tasks



The user already has a mental model that describes the task your software is enabling. This model arises from a combination of real-world experiences, experience with other software, and with computers in general. For example, users have real-world experience writing and mailing letters and most users have used email applications to write and send email. Based on this, a user has a conceptual model of this task that includes certain expectations, such as the ability to create a new letter, select a recipient, and send the letter. An email application that ignores the user's mental model and does not meet at least some of the user's expectations would be difficult and even unpleasant to use. This is because such an application imposes an unfamiliar conceptual model on its users instead of building on the knowledge and experiences those users already have.

Before you design your application's user interface, try to discover your users' mental model of the task your application helps them perform. Be aware of the model's inherent metaphors, which represent conceptual components of the task.

The mental model your users have should infuse the design of your application's user interface. It should inform the layout of your application's windows, the selection and organization of icons and controls in the toolbars, and the functionality of utility windows.

Users should not have to learn new things to perform familiar tasks. The use of concepts and techniques that users already understand from their real world experiences allows them to get started quickly and make progress immediately.

By choosing to be consistent with something the user already understands, an interface can be made easier to learn, more productive, and even fun to use. Providing a familiar experience is the ultimate use of consistency in which a truly intuitive interface will result.



4. Keep the User Informed



Always let the user know what is happening in your application by using appropriate feedback at an appropriate time. The user should never have to guess about the status of the system or of your application. When the user performs an action, provide feedback to indicate that the system has received the input and is operating on it.

For potentially lengthy operations, use a progress indicator to provide useful information about how long the operation will take. Users don't need to know precisely how many seconds an operation will take, but an estimate is helpful.

The system should keep track of:

- ❑ Whether this is the first time the user has been in the system
- ❑ Where the user is
- ❑ Where the user is going
- ❑ Where the user has been during this session
- ❑ Where the user was when they left off in the last session

and myriad other details.

Although highly responsive applications can differ widely from one another, they share the following characteristics:

- ❑ They give immediate feedback to users, even when they cannot fulfil their requests immediately
- ❑ They provide enough feedback for users to understand what they are doing, and organize feedback according to users' abilities to comprehend and react to it
- ❑ They let users know when processing is in progress
- ❑ They let users know or estimate how long lengthy operations will take

Highly responsive applications put users in control by quickly acknowledging each user request, by providing continuous feedback about progress toward fulfilling each request.



5. Provide appropriate feedback



It is critical that feedback be *accurate* and *precise*. If you display a determinate progress indicator to display the state of completion of a task and it is inaccurate, the user will lose faith in progress indicators, and they will find the environment less usable. If you display a generic error message that indicates that there is a problem but fails to provide enough information to diagnose or solve the problem, your users will be unable to continue with their task.

When a user initiates an action, always provide an indication that your application has received the user's input and is operating on it.

Users want to know that a command is being carried out. If a command can't be carried out, they want to know why it can't and what can be done instead.

A good warning or error message contains the following elements:

1. A brief description of the problem
2. A list of ways the user can remedy the problem

Both of these elements should be presented in simple, non-technical and jargon-free language.

6. Put the User in Control



As much as possible, allow users to do whatever they want at all times. Users should always feel in control, able to do what they want when they want. Users should be able to switch between different tasks at any time. Avoid using interfaces that lock them into one operation and prevent them from switching to anything else until that operation is completed. Users should always have a clear path out. Avoid interfaces that make users feel trapped. Use stable visual elements to enable people to navigate fast but also to allow them have dependable landmarks, giving people a sense of control on the navigation.



Try to use accelerator elements in a manner that they are unseen to novice users but speed up the interaction for the expert users so that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

To give users control over the system, enable them to accomplish tasks using any sequence of steps that they would naturally use. Don't limit them by artificially restricting their choices to your notion of the "correct" sequence.

The user should also be able to tailor aspects of their environment to fit personal preferences. It is very important, however, to avoid the trap of allowing too much configuration, or allowing the configuration of parameters that most users will not understand or find useful to modify.

It's important to provide users with the capabilities they need while helping them avoid dangerous, irreversible actions. For example, in situations where the user might destroy data accidentally, you should always provide a warning, but allow the user to proceed if they choose.

7. Make Actions predictable and reversible



Encourage people to explore your application by building in forgiveness—that is, making most actions easily reversible. People explore in ways beyond navigation. Sometimes they want to find out what would happen if they carried out some potentially dangerous action. Sometimes they don't want to find out, but they do anyway by accident.

Users should feel confident in exploring, knowing they can try an action, view the result, and undo the action if the result is unacceptable. They feel more comfortable with interfaces in which their actions do not cause irreversible consequences.

Studies showed that people in a hazardous environment make no more mistakes than people in a supportive and more visually obvious environment, but they worked a lot slower and a lot more carefully to avoid making errors.

Ensure that users never lose their work as a result of error on their part, the vagaries of Internet transmission, or any other unavoidable reason. If an action is very dangerous, and there is no way to undo the result, warn the user and ask for confirmation. Only do this in



extreme cases, though; if frequently faced with such confirmation messages, users begin to ignore them, making them worse than useless.

Even seemingly trivial user actions, such as deselection or moving objects, should be reversible. For example, a user who spends several minutes deliberating and selecting individual files to be archived from a group will be very upset if all the files are accidentally deselected and the deselection cannot be undone.

Avoid bundling actions together, because the user may not anticipate the side effect. For example, if a user chooses to cancel a request to send a note, only the send request should be cancelled. Do not bundle another action, such as deletion of the note, with the cancel request. Rather than implementing composite actions, make actions independent and provide ways to allow users to combine them when they wish.

In all cases, nothing your application does should lose or destroy user's work without explicit user's will. Among other techniques, this can be achieved by autosaving backups of documents, and allowing multiple levels of undo.

8. Reduce latency



Reduce the user's experience of latency.

- ❑ Acknowledge all button clicks by visual feedback
- ❑ Display a message indicating the potential length of relatively long processes
- ❑ Communicate the actual length through an animated progress indicator

Not knowing how long to wait for a page to be loaded or an action to be performed is often very frustrating to users. In fact, Hui and Tse (1996) found that users reported negative effects on mood if they had no information concerning how long they have to wait, which affected their overall evaluation. Thus, for longer waits it is recommended to inform the user how long the wait should generally take.

Make each response delay in your application as short as possible. The acceptable response delay for each event is based on a typical user's sense of the potential length of the event. The greater that sense is, the more willingly the user will wait for a response.

For example, the acceptable time needed for a web page to be opened is generally not more than 8 seconds over a 56 kbps modem. Verify that your application provides feedback within



an acceptable time after each key press, movement of the mouse, or other physical input from the user. The feedback for inherently long tasks can be a progress indicator.

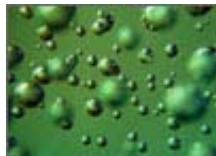
9. Avoid feature bloat



It's true that softwares are expected to evolve in order to answer upcoming needs of their audience. But evolution of the software does not only mean addition of new features. It also means improving the stability, usability and accessibility of the existing features. Softwares that focus only on adding new features become soon confusing and hardly stable. Remember that releasing a useful software requires a careful balancing of innovation and stability.

Focus on solutions, not features. Avoid adding features solely for competitive reasons. Make sure every feature offers real benefit to your target users. Recognize the cost of each feature you add in terms of complexity of the software, usability and accessibility of the software, focused user switch, etc.

10. Manage Complexity



Keep the interface simple and straightforward. Users benefit from function that is easily accessible and usable. A poorly organized interface cluttered with many advanced functions distracts users from accomplishing their everyday tasks. A well-organized interface that supports the user's tasks fades into the background and allows the user to work efficiently. Basic functions should be immediately apparent, while advanced functions may be less obvious to new users. Function should be included only if a task analysis shows it is needed.

Every extra piece of information or interface control competes with the truly relevant bits of information and distracts the user from important information. Hence, don't clutter your



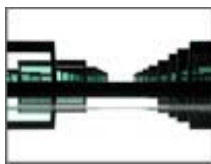
interface, and don't overload the user with buttons, menu options, icons, or irrelevant information.

The more you can do to simplify the interface of your application for your users, the more likely it is that you will build a product that meets their needs and is enjoyable to use. The more complex your application's task, the more important it is to keep the user interface simple and focused.

In addition to creating a streamlined design, you can also manage complexity in the following ways:

- ❑ Progressive disclosure presents the most common choices to the user first and provides an option that allows the user to view additional information and choices. This technique makes it easy for novice users to understand your user interface while still giving power users the advanced features they want.
- ❑ Info windows and inspector windows reduce the clutter of a user interface by placing additional information and settings in a separate window that can be hidden or shown by the user.
- ❑ Preferences reduce the complexity of the user interface by giving users the ability to customize what they see on the display screen and, to some extent, how the application performs. By providing preferences, you allow both novice and expert users to mould your application to fit their needs.

11. Make Your Application Consistent



Make your application consistent in its appearance, behaviour, visual concepts and verbal language. Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

The unity of the interface is important because it has the potential to link concepts and objects together that belong together. For example, Wickens' (1986) compatibility of proximity principle states that tasks that necessitate mental integration of information should be in close proximity. This can be applied to things such as the organization of links. For instance, care should be made to group links that belong together, as well as separate those that do not belong.



Consistency in the interface allows users to transfer their knowledge and skills from one application to another. You can positively extend the consistency principal by asking yourself the following questions when developing your application:

- ❑ Is it consistent with the user interface guidelines? Review the guidelines to make sure they have been respected in your application.
- ❑ Is it consistent within itself? Does it use consistent terminology for labels and features? Do icons mean the same thing every time they are used? Are concepts presented in similar ways across all modules? Are similar controls and other user interface elements located in similar places in windows and dialogs?
- ❑ Is it consistent with earlier versions of the product? Have the terms and meanings remained the same between releases? Are the fundamental concepts essentially unchanged?
- ❑ Is it consistent with people's expectations? Does it meet the needs of the user without extraneous features? Does it conform to the user's prior knowledge?

Balance and unity has always been a key component in good design. Humans on a preconscious level seek structure in the things they see. If there is no intentional structure, we will impose our own. Seeking the appropriate balance among things, as well as unifying those things that are related will generate structures that are not only pleasing to the eye, but will make the interface more understandable.

12. Provide an aesthetic Integrity



Aesthetic integrity means that information is well organized and consistent with principles of good visual design. Your product should look pleasant on the screen, even when viewed for a long time.

Visual design is not just about making your application look pretty. Good visual design is about communication. A well-designed application will make it easy for the user to understand the information that is being presented, and show them clearly how they can interact with that information.

The following are visual design principles that promote clarity in the interface:



❑ **Subtractive design**

Keep graphics simple, and use them only when they truly enhance usability. Don't overload windows and dialogs with dozens of icons or buttons. Reduce clutter by eliminating any visual element that doesn't contribute directly to visual communication.

❑ **Visual hierarchy**

By understanding the importance of users' tasks, establish a hierarchy of these tasks visually. An important object can be given extra visual prominence. Relative position and contrast in colour and size can be used.

❑ **Extra space**

Do not eliminate extra space in your image just to save space. Use white space to provide visual "breathing room."

❑ **Icons**

Icons can assist the user in rapidly scanning a large number of objects to select the desired item. Particularly after a user is accustomed to an icon's appearance, they can identify it more rapidly than a text label.

❑ **Fonts**

Do not mix more than two or three font sizes and styles in one window, as this will look unprofessional and distract the user from the information being conveyed.

❑ **Colours**

Colour is a good tool for communicating information in a user interface. However, colour should always be regarded as a useful addition to your design, not as a necessity. Never depend upon colours alone to display important information, and keep in mind that if colours cannot be perceived correctly for example, if the user has an 8-bit system, or if the user is colour-blind, your application should still be usable.



13. Allow users to customize



The interface should be tailorable to individual users' needs and desires. No two users are exactly alike. Users have varying backgrounds, interests, motivations, levels of experience, and physical abilities. Customisation can help make an interface feel comfortable and familiar.

Personalizing the interface can also lead to higher productivity and user satisfaction allowing users to save time and effort when accessing frequently used functions.

14. Make an “always at the same place” illusion



Most users cannot and will not build elaborate mental maps and will become lost or tired if expected to do so. Present the illusion that users are always in the same place, with the work brought to them. This not only eliminates the need for maps and other navigational aids, it offers users a greater sense of mastery and autonomy.

Minimize the user's memory load by making objects, actions, and options always visible and at the same place. The user should not have to remember information from one part of the dialogue to another.



15. Support Direct Manipulation of objects



Direct manipulation is an action that allows users to feel that they are controlling the objects represented by the computer. According to this principle, an onscreen object should remain visible while a user performs an action on it, and the impact of the action should be immediately visible. For example, with a drag-and-drop operation (the most common example of direct manipulation) users can move a file by dragging its icon from one location to another, or drag selected text directly into another document. Another example of direct manipulation is the resizing of a graphic object in a drawing application.

Support direct manipulation when users are likely to expect it. Avoid forcing users to use controls to manipulate data. Make sure there are no significant differences between what users do onscreen and what they receive in the output. When the user makes changes to a document, display the results immediately; the user shouldn't have to wait for the final output or make mental calculations about how the document will look later. Use a preview function if necessary.

16. Shorten labels with the key word(s) first



Menu and button labels should be precise have the key word(s) first.

Look at the following examples borrowed from a word processor:

Bad labels:

- Insert page break
- Add Footnote
- Update Table of Contents

Good labels:



Insert:

- ❑ Page break
- ❑ Footnote
- ❑ Table of contents

17. Reduce the distance of successively used elements



The time to acquire a target is a function of the distance to and size of the target.

While at first glance, this law might seem patently obvious, it is one of the most ignored principles in design. If there are several different buttons, links or other objects that need to be clicked in succession in order to perform a specific task, the smaller and the farther apart they are, the longer it will take to perform the task and consequently the more frustrating the process will be especially when it needs to be repeated several times.

18. Prevent errors



Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

Anticipate common problems and provide extensive feedback and communication at every stage to protect users from making errors by helping them make the right choices. The burden of keeping the user out of trouble rests on the designer. The interface should provide visual cues, reminders, lists of choices, and other aids, either automatically or on request. Humans are much better at recognition than recall. Contextual and hover help, as well as



agents, can provide supplemental assistance. Simply stated, eliminate the opportunity for user error and confusion.

19. Provide help and documentation



Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

Users should never have to rely on their own memory for something the system already knows, such as previous settings, file names, and other interface details. If the information is in the system in any form, the system should provide it.

Two-way communication may be necessary at times to allow users to clarify or confirm requests, or to remedy a problem. The communication should be interactive - as rich in presentation and interaction capabilities as the rest of the interface. It should present relevant information, provide access to related information and help, and allow users to make task-specific decisions to continue.

Adopt the following design perspective: users know what they want to accomplish, but sometimes they find it difficult to express their desires using the objects and actions provided, and the system is unable to recognize their request. Two-way communication may be used to help users reach their goals.

Most users perform a variety of tasks, being expert at some and novice at others. In addition to providing assistance when requested, the system should recognize and anticipate the user's goals, and offer assistance to make the task easier. Ideally, assistance should provide users with knowledge that will allow them to accomplish their tasks quickly. Intelligent assistance is like the training wheels on a bicycle - at some point, most users will want to take them off and go forward on their own. The assistance should allow them to become independent at some point when they choose to be so.



20. Be aware of cultural differences



Make sure that visible interface elements can be localized, that is, translated into other languages and otherwise adapted for use in other countries.

Sensitivity to cultural and political issues is an important consideration. Designing icons and even choosing colours requires some understanding of the connotations they might have to a user from a different part of the world. Examples of elements it is best to avoid for these reasons include:

- ❑ Pictures of flags or money
- ❑ Maps showing political boundaries or contentious location names
- ❑ Lists of countries or cities in non-alphabetical order (unless specifically requested or required by the context)
- ❑ Icons depicting animals
- ❑ Icons depicting gestures of hands, feet or face

Certain images can be normal or even pleasant in some cultures but offensive to certain other ones. Cultures assign varying values and characteristics to living creatures, plants, and inanimate objects. For example, in the United States and Europe the owl is generally a symbol of wisdom and knowledge, whereas in Central America or in Middle East the owl represents witchcraft and black magic.

It's a good idea to avoid the use of culturally different concepts like holiday names or calendar events in software that you expect to distribute worldwide unless you localize the concept for all cultures. Remember that different calendars are used to mark time around the world. Except the Christian calendar used in most of European countries and in the Unites States, there are others calendars such as Chinese calendar, Indian calendar, Ethiopian calendar, Jewish calendar, Persian calendar and Islamic calendar. In many places, time is marked according to one calendar for business and government purposes, and another for religious events.

Also remember to support different address formats. Don't assume all addresses are in the native format of your country. Be prepared to handle different address formats and postal code information.



21. Be aware of linguistic differences



Translating text is a sophisticated, delicate task. Avoid using colloquial phrases or non-standard usage and syntax. Carefully choose words for menu commands, dialogs, and help text.

Use complete sentences in string resources whenever possible. Grammar problems may arise when you concatenate multiple strings to create sentences. The basic order of subject, verb, and object in sentences in different languages is completely different rendering multiple string sentences nonsensical when translated. For example languages like English, Chinese, Romance languages and Russian follow the SVO (Subject, Verb, Object) pattern while languages like Turkish, Japanese, Persian and most Indian languages follow SOV (Subject, Object, Verb) pattern and languages like Irish, Scottish Gaelic, Tagalog and Tongan use the VSO (Verb, Subject, Object) pattern.

Writing systems differ in the direction in which their characters and lines flow, the size of the character set used, and whether certain characters are context-dependent. Be sure that your application support Unicode in order to support the character encoding of most writing systems in the world. When creating window layouts, consider text size, location, and direction.

No matter what level of worldwide text support you provide, it's important to keep in mind that:

- ❑ Text isn't always left-aligned and read from left to right.
- ❑ Text isn't always read by a person; it might be spoken through a screen reader.
- ❑ System and application fonts may change, so don't assume any particular font will be present. Instead, use the calls provided by your application framework.



22. Don't exclude people with special needs



Millions of people have a disability or special need and computers and softwares hold tremendous promise for providing access to such users. It's a good idea to build in support for universal access from the beginning of your design process rather than add it at the end of your implementation cycle. Be sure to test your applications with the assistive features for people with special needs. When you design your application, you should be aware of potential manipulation by assistive technologies and implement features in a way that does not degrade the experience for users with disabilities.

On the other hand, you should be careful not to overcompensate for the disabilities of certain members. Don't let accommodations for a particular disability create a burden for people who do not have that disability.

The following sections describe the main categories of disabilities and give suggestions for specific design solutions and adaptations you can make. Keep in mind that there is a wide range of disabilities within each category and there are many people with multiple disabilities.

Visual Disabilities

Users with vision disorders, such as colour-blindness or low vision, require alternatives to default settings. A good user interface anticipates these needs by providing customisable preferences and support for accessible themes. Even better is an application that is already configured with carefully- chosen colour and contrast defaults.

Pay particular attention to the needs of older people. Presbyopia, the condition of hardened, less flexible lenses, coupled with reduced light transmission into the eye, affects most people over age 45. Do not trust your young eyes to make size and contrast decisions.

An estimated 11% of the world population has some sort of colour-blindness. Those affected typically have trouble distinguishing between certain hues such as red and green (deuteranopia or protanopia), or blue and yellow (tritanopia). Therefore it is necessary to allow the user to customize colours in any part of your application that conveys important information. This means that your application must effectively convey information using just the colours from any theme that the user chooses.

A useful tool for reviewing information about colour-blindness and checking legibility of images for colour-blind users is Vischeck [<http://www.vischeck.com/>], an online tool that simulates the way an image or a website might appear to a user who has deuteranopia, protanopia, or tritanopia.



Other users have more problems with contrast levels rather than hue on their screen. Some users require a high level of contrast between background and foreground colours, such as black on white, white on black, or some other high-contrast combination. Others can experience discomfort unless they use low-contrast settings, such as grey text on a lighter grey background.

Dokeos can meet these needs by ensuring the css and theme customisation possibility for these users. Take into account the following to ensure that your software supports people with visual disabilities:

- ❑ Use alternative texts and description for images, graphs and charts
- ❑ Use meaningful names and "NOFRAME" alternatives for frames
- ❑ Use relative font sizes that can be enlarged or reduced easily
- ❑ Don't use tables that do not make sense when read serially (in a cell-by-cell or "linearised" mode)
- ❑ Provide alternative access keys for forms and make sure they can be tabbed through in a logical sequence
- ❑ Don't use text presented as images even if you use alternative text because they can prevent wrapping to the next line when enlarged
- ❑ Provide large sizes of high-, low- and regular contrast icon sets for people with visual disabilities
- ❑ Ensure your application is not dependent on a particular theme. Provide different themes, especially high and low contrast accessibility themes
- ❑ Select colours carefully. When they need to be recognizably different, select the light colours from orange, yellow, green or blue-green, and darker colours from blue, violet, purple or red, as most people affected by colour-blindness already see blue, violet, purple and red as darker than normal.

Other disabilities

Other forms of disabilities that should be taken into account in a user interface design include motor disabilities, visual and auditory perceptual disabilities, attention deficit disorder, intellectual impairments, memory impairments, mental health disabilities, seizure disorders, hearing impairments (deafness, hard of hearing) and aging-related conditions.

In order to remove the barriers that people with these disabilities may encounter, pay attention to the following when you design your application:

- ❑ Use relative font sizes and alternative text for visuals
- ❑ Use captions for audio and video
- ❑ Don't use distracting visual or audio elements that cannot easily be turned off
- ❑ Use a simple and clear navigation structure
- ❑ Add appropriate graphics to text labels to help people identify them easier



23. Be conformant to public norms



Market typically drives the various leaders from business, academia, and government to work together to develop common “norms.” This convergence of technologies is very important for the consumers of these technologies because your adherence to public norms will ensure consumers that your product will avoid quick obsolescence, will work in any context and is more open to possible changes.

E-learning addresses wide populations in various countries using various softwares in various contexts. Courses, Learning Management Systems and all kinds of learning-oriented devices should be usable in any browser, translatable in any language, importable in any Learning Management System, CD-Rom or textbook.

Make sure that you respect common e-learning norms for things such as content metadata, content packaging, content sequencing, question and test interoperability, learner profiles, run-time interaction, etc. and also common web norms for accessibility and interoperability issues.



References

- BENSON, C., ELMAN, A., NICKELL, S., ROBERTSON, C., "GNOME Human Interface Guidelines 2.0"
<http://developer.gnome.org/projects/gup/hig/2.0/hig-2.0.pdf>
- BIOT, S., 2002. "KDE Usability — First Steps," KDE Usability Project
<http://usability.kde.org/activity/testing/firststeps/>
- DE PRAETERE, T., "The Dokeos 10 steps quality management"
<http://www.dokeos.com/quality.php>
- EKLUND, S., FELDMAN, M., TROMBLEY, M., SINHA, R., "Improving the Usability of Open Source Software"
<http://www.sims.berkeley.edu/~sinha/opensource.html>
- NICHOLS, D. M., TWIDALE, M. B., "The Usability of Open Source Software"
http://www.firstmonday.dk/issues/issue8_1/nichols/#author
- NICKELL, S., 2001. "Why GNOME Hackers Should Care about Usability,"
http://developer.gnome.org/projects/gup/articles/why_care/
- NICKELL, S., "Improving Usability: Principles and Steps for Better Software"
<http://www.redhat.com/magazine/002dec04/features/usability/>
- NIELSON, J., "Ten Usability Heuristics"
<http://www.useit.com/jakob/>
- RAYMOND, E., S., 1998. "The Cathedral and the Bazaar,"
http://firstmonday.org/issues/issue3_3/raymond/
- TALIN, "A Summary of Principles for User-Interface Design"
http://www.sylvantech.com/~talin/projects/ui_design.html
- TOGNAZZINI, B., "First Principles of Interaction Design"
<http://www.asktog.com/basics/firstPrinciples.html>
- "Apple Human Interface Guidelines"
<http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/>
- "IBM Ease of Use"
http://www-306.ibm.com/ibm/easy/eou_ext.nsf/publish/558
- KDE Usability Project, "KDE Usability"
<http://usability.kde.org>
- MACIE Centre, "Making Sense of Learning Specifications & Standards"
http://www.masie.com/standards/s3_2nd_edition.pdf
- W3C Working Draft, 10 December 2004, "How People with Disabilities Use the Web"
<http://www.w3.org/WAI/EO/Drafts/PWD-Use-Web/Overview.html>