User Experience Direct
(UX Direct)

FAQ: How to conduct Heuristic Evaluation

Disclaimer

FAQ: How to conduct Heuristic Evaluation

User Experience Direct (UX Direct) is an Oracle Applications User Experience (UX) program that provides user experience expertise to Oracle customers and partners for their implementations, customizations, and usage of Oracle enterprise applications. The goal of this program is to enhance end user experiences during and after customer implementations and improve user adoption of Oracle's enterprise applications.

Overview

A heuristic evaluation is a systematic inspection of a user interface to uncover as many of the problems users might have as time allows. The people doing the inspection perform typical tasks that users would want to do with the product and record the problems that they expect users will have. The problems are violations of the 10 or so heuristics that form the basis of the inspection. This document describes how to perform a heuristic evaluation.

1. What are the heuristics and where do they come from?

The 10 heuristics that are most commonly used in user interface inspections were developed from research conducted in the 1990s. They are:

1. Visibility of feedback
2. Complexity of the application
3. Task navigation and user controls
4. Consistency and standards
5. Error prevention and correction
6. Recognition rather than memory overload
7. Efficient to use
8. Simplicity and appeal
9. Be tolerant and reduce cost of errors
10. Help Support

A complete description of these heuristics, with examples, can be found in Appendix A and in the User Experience Direct: User Interface Heuristics Checklist collateral.

2. Who conducts a heuristic evaluation?

Ideally, a user experience (UX) professional conducts the evaluation because he or she has been trained in the method. Research has shown that UX professionals find the most problems.

If no UX professional is available, people from other professions can use the method. One way to enhance the reliability of the method is to have several people, say three to five, conduct the evaluation independently and then get together to consolidate their lists of problems into one common list.

3. Can the method be used for an implementation?

Yes. This method is used both during the initial development of an application and during the configuration and customization of an application.

4. What are the advantages of a heuristic evaluation?

There are two primary advantages. First, it can be conducted quickly. The whole process can be completed in a couple of days. Second, it is efficient. It is sometimes called a "discount" method because it requires fewer resources than other methods.

5. What are the disadvantages?

The primary disadvantage is that the method does not involve end users. Consequently, some of the problems that are identified may be dismissed by stakeholders because they are "just the opinion" of the people who conducted the inspection.

One of the ways to deal with this dismissal is to make the process of the evaluation explicit. It is not based on opinion but on a systematic inspection of steps that users follow to complete tasks. Show how the method works and explain the problems with screen shots that illustrate them.

6. How do I prepare for a heuristic evaluation?

Follow these steps to prepare:

- Decide on the scope of the inspection. Will it be applied to the whole product? To the sections that have been customized?
- Identify the roles and characteristics of the users who will be using the product.
- Make a list of the tasks that users will perform frequently with the product.

- Create a form that evaluators will use to record the problems they see, showing on which page of the software it occurred. (See Appendix B for a sample form.)
- Make the product or screen shots of it available to those who will inspect it.

7. How do I conduct the heuristic evaluation?

The method described here assumes that a UX professional is not available to the design team and that more than one evaluator will conduct the inspection.

- The evaluators are briefed on the scope of the inspection and the characteristics of the users who will be using the product. They review the list of 10 heuristics and examples to ensure that they understand them.
- The evaluators then independently work through the tasks selected for the inspection. Evaluators take the point of view of the intended users of the product. When evaluators uncover a problem that they believe users will have, they record it on the form along with the task attempted, the page or dialog that is displayed at the time, and the heuristic that it violates. For example, a field requiring the typing of a date does not show the valid format for the date. The evaluator records the problem, the page, and field on which it occurs and the heuristic: Error Prevention and Correction. In another example, after the user clicks on a Submit button, there is a long (5-second) pause before the page refreshes. The evaluator records the problem, the page on which it occurs, and the heuristic: Visibility of Feedback. (See Appendix B for a sample form.)
- After the evaluators complete the tasks selected for the inspection, they should explore the product in other ways. They might try to create error conditions and explore pages not encountered during the tasks.
- After the inspection of the product is completed, each evaluator goes back to the forms and records any obvious solutions to the problems they uncovered. For example, an evaluator might suggest providing a prompt that shows the format of a date or displaying a processing indicator when there is more than a two-second pause. Research has shown that it is better to think about solutions after the problems are recorded rather than at the same time they are encountered.
- Finally, the evaluators get together and go over their problem sheets and solutions, combining their problems and eliminating duplicate problems.

8. Are all usability problems treated equally?

No. Some problems are more important than others. There are at least two ways in which problems differ. Problems may impact users differently. For example, will the problem cause users to fail to complete a task, or does the task take much longer to complete that users expected? On the other end of severity, the problem may be a minor misunderstanding that only shows up the first time the product is used. Also, some problems are likely to affect every user, while others affect only a small subset of users.

The second aspect to consider is the scope of the problem. Does it affect only one screen, or does it have a wider impact? For example, structural issues such as the organization of tabs or menus that do not match the users' work flow may affect more than one page. However, a single misunderstood term is likely to affect just one step within a task.

A simple but effective way to categorize the severity of problems is to use a three-level scheme: high, medium, and low severity. A person who has inspected the product can make this judgment, and it will not be much different from the categories of a UX expert.

It may also be helpful to identify problems that have solutions that are easy to accomplish. For example, you can say: "Here is a list of problems that can be fixed with little expenditure of resources."

9. How do I present the results of a heuristic evaluation?

Because the problem lists that result from an inspection are based on professional judgment rather than from the performance or opinions of users, it is important to make the results credible. If you are presenting the results to managers who have to make decisions about what to fix, keep in mind that they were not present to watch the evaluation process. They may have no idea that the problems were identified by a systematic process. Consequently, it is important to describe the heuristic evaluation process, show the heuristics, and show the forms used to record them.

A second strategy when presenting the results is to avoid overwhelming the audience with a very long list of problems. Many of the problems may be minor, low-priority issues. This is especially true if the product has not been subjected to a quality assurance check. One way to deal with this is to separate the top 10 problems from the rest and to focus on them. In most cases, if you can get the organization to fix only those top 10 issues, the product's usability will be significantly enhanced.

Appendix A: User Interface Heuristics Checklist

Overview

Usability heuristics are general principles that help guide user interface design. The list below describes the principles Oracle uses to create strong, usable designs that offer a best-in-class user experience. The heuristics also can be used to guide an organization's configuration and customization of Oracle applications.

1. Visibility of system feedback:

Does the application keep users informed about what is going on through visible, clear, and concise feedback? The application should provide indicators to answer the question, "Where am I?"

Design Perspective:

The interaction of a user with an application is, in many ways, like the interaction of two people talking on the phone. The two people can't see each other, so use verbal cues and sounds to aid the communication. Here is an example:

He: Wait, I will get a pencil. (Statement of action.)

She: OK, I'll wait. (Feedback that she heard the request.)

(Note: if the pause on the phone is more than a few seconds, she will likely say, "Are you still there?" because there is no indication that the line is still active and that he is OK.)

He: OK. Go ahead. (Feedback that he is back and passing control of the conversation.)

She: The name is JANE DOE. (Passing information.)

He: JANEDOE. (Quietly mimicking her as she says the letters – feedback that he is hearing the letters correctly but not taking control of the conversation.)

He: OK. (Indicating that he has the name and inviting her to keep control.)

She: 17 ABBEY. (Passing information.)

He: Wait! Was that EY? (Error message that takes control of the conversation and asks for confirmation.)

She: Yes. (Verification and taking back control of the conversation.)

She: In Watertown…did you get that? (Passing information and asking for confirmation.)

He: Yes, give me the zip. (Confirmation and request for information; passing control back.)

She: I am looking it up …let's see… here it is … 09873. (Statement of action, verbal feedback indicating both that she is working at the task and still in control of the conversation. Then passing information.)

You can see that throughout this conversation, there is a constant passing of control, continuous feedback, and verification about the status of the conversation and the information being passed. Long pauses cause either party to question whether the conversation is still active, and sometimes subtle cues are used to keep control of the conversation or to indicate that the other party should continue.

In human-computer interactions, there are visual analogs to the verbal cues used in conversation. The user can only see what is on the current screen. So the user needs indications about where this screen is and what it is about. Titles and headers on a screen are essential to knowing what it contains. Breadcrumbs help by indicating where the screen is in the structure of the application. In screen sequences, such as wizards and trains, messages like "Step 2 of 4" provide status feedback.

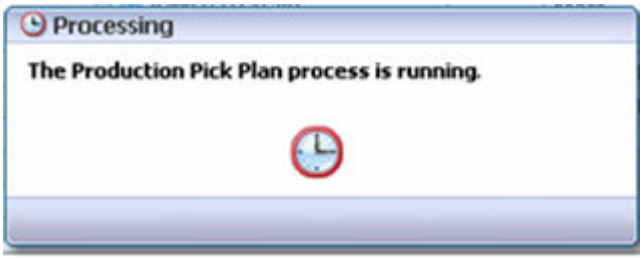
With enterprise applications, the core functionality is provided by the core software, which ideally has been reviewed and tested with users to ensure it provided the proper feedback. Just as important is the design of the additions to the core functions made by the customer's implementation team. Configurations and customizations also need to follow the same good practices about feedback.

When users have taken some action, such as clicking on a button, they need feedback about the actions the application is taking in response. Sometimes moving to a new page is enough. But often the user needs more, such as a status message saying that the database has been updated. This verifies to the user that the action he or she performed had the desired result. The message can be a simple pop-up message that requires no action, which is analogous to providing feedback without taking control of the conversation.

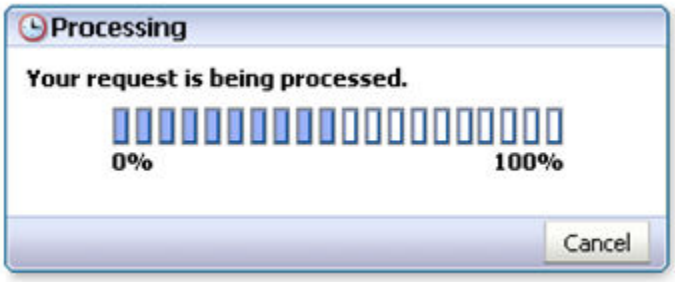
In human-computer interaction, it often happens that the computer drops out of the conversation unexpectedly. It happens enough that most users begin to suspect it has happened after only a few seconds, unless the application provides feedback in the form of progress indicators. For short pauses, a simple symbol such as an hourglass cursor is enough. But for pauses of more than about 5 seconds, a progress bar can keep users from trying to see if the application is active by pressing the enter key several times.

Examples of good practices:

1. Indicate where the user is with a title or header on every page.
2. In multi-page processes, indicate progress. For example: “Step 2 of 4.”
3. Indicate that the application is working normally with an icon that appears when the application is processing, to prevent the user from performing any action.



For longer operations, use a progress indicator to show how much of the process is left to complete.



4. Indicate when a task is successfully completed.

5. Indicate the path through an application showing “breadcrumbs" at the top of the page. For example: “Suppliers: Address Book>Create Address>Confirm Details.”

2. Complexity of the application:

Does the application match the user's real world and language, and reflect the user's primary goals and tasks? The application should speak the user's language, with familiar words and phrases, and organize information in a natural and logical order.

Design Perspective:

One of the biggest challenges for an application designer is understanding the users' world -- their language, the tasks they perform, and the way they structure their work. Users of enterprise software are experts at their jobs. They have specialized degrees, training, and varied experience. Ideally, the core application has been designed around users and their work. But in the process of implementing an application at a company, the users' characteristics must also be considered. Configurations and customizations need to speak a user's language and be organized around a user's work practices, which can only happen when users are involved in the design of these additions as part of the design team or as evaluators during usability tests.

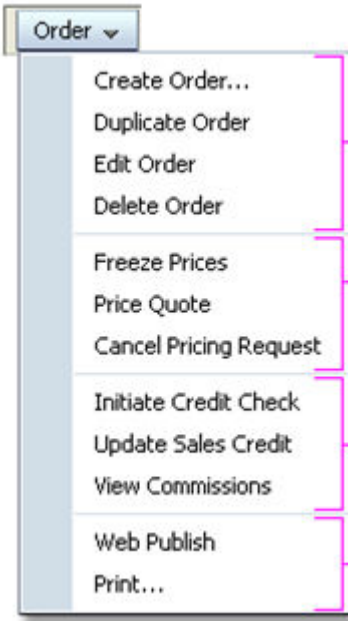
Most applications are organized through tabs, sub-tabs, and pull-down menus. Designers often think of partitioning an application into the functions it performs. But these menus and tabs work best when the terminology and organization of the design fits a user's mental model of work. Conducting interviews with users and observing them is essential to getting the task flow right. One of the important challenges to user-centered design comes when new functions are implemented or when a new application or version changes the way work is done. In those cases, designers can't look backward to see how work was done but must look forward by working with users to evaluate a design that introduces change. In these cases, it is not sufficient to ask users what they expect. Users need to see the screens and use the evolving design to ensure its usability.

Card sorting has proven to be a useful way to organize lists, menus, and tabs. One way this method is used is to ask potential users to review the names of options that the designers suggest and to propose their own terms. Then they are asked to sort the options into piles and name the piles. The results of this sorting can be used as suggested ways to organize the application.

Examples of good practices:

1. Make buttons look as if they have been pushed when the user clicks on them.

2. Organize menu choices in the most logical way for the user, the item names, and the task.



3. Place related and inter-dependent fields on the same page or section of a page.
4. Use prompts in data-entry fields, such as "MM/DD/YYYY."
5. Create buttons and links that employ user jargon but avoid computer or technical jargon.

3. Task navigation and user controls:

Does the application allow users to complete tasks and goals without hindrance? The application should allow users to undo actions.

Design Perspective:

In addition to terminology and structure, navigation is the third leg of a usable design. Applications provide users with a limited number of paths to achieve their goals. In a Web-based design, there is a tension between simplicity and flexibility. The more paths that are possible, the more flexible the design. But flexibility comes at a cost because it increases complexity. The ultimate in flexibility can be achieved by providing no structure, allowing users to do whatever they want. But such a design is nearly impossible to learn to use. The key to an effective design is to balance flexibility and simplicity.

A design principle that speaks to this issue is to keep the user in the context of his or her work as much as possible. Doing so requires an intimate knowledge of the work process. Work is broken down into steps. Within a step, the design should keep users in context as much as possible by not sending them off to new pages and then bringing them back later. In the early history of HTML interfaces, the server did not know what the user was doing until an action triggered a hit to the server, and it was impossible to refresh only the part of the page the user was on. Those restrictions have changed. For example, when the user takes an action that brings up a table of data, the table can be inserted into the current page, leaving the context intact. In addition, tables can have controls on them that allow users to move to the next step without leaving the table or the page. Pop-up dialogs can sit on top of the page so that they only cover part of it, allowing users to make decisions or add new data while staying in the context of their work.

One of the characteristics of people is that we make mistakes. We click without thinking. And we change our minds. One of the most frustrating aspects of using an application is not being able to recover from an action we didn't intend or not being able to reconsider and go back a step. When designing a configuration or customization of a core application, it is important to add "Undo" and "Redo" options whenever possible. It takes extra work, but is enhances usability. Also, in sequential processes, users should be able to move back and forth through the steps until they are finished. Sequential processes are usually designed to guide inexperienced users through the steps in a process, which makes free movement

even more necessary as they are likely to make missteps or change their minds.

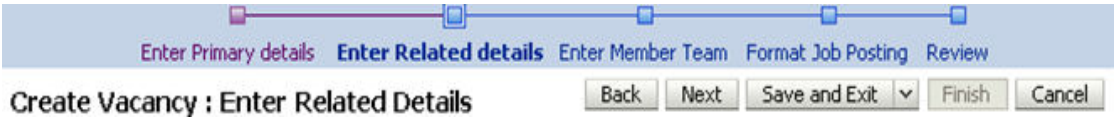
Traditionally, navigation buttons performed a single function, such as “Save” or “Submit,” which usually then brought users to a new page. But navigation buttons can have multiple functions when those functions frequently occur together, such as “Save” and “Add Another” or “Edit and Submit.” These types of buttons reduce the number of jumps to new pages and keep users in the context of their work longer.

Examples of good practices:

1. Match the flow of pages to the flow of user tasks.
2. Allow users to confirm actions that have drastic or destructive consequences, for example:

You have unsaved changes. Save changes before continuing? Yes/No

3. Allow users to reduce data-entry time by copying and modifying existing data.
4. Allow users to easily undo actions, for example: "Remove from cart."
5. In multi-page tasks, allow users to move backward and forward among all the pages in the set.



4. Consistency and standards:

Does the application follow Web-based or product-based standards and conventions consistently? Users should not have to wonder whether different words, situations, or actions mean the same thing.

Design Perspective:

Creating a usable application is a group process. Different people work on different parts of the user interface. In order to ensure the application’s usability, all of these people need to be following the same conventions. In addition, when an application is being configured and customized, the people working on the user interface need to following the same conventions as the core application. For example, all Save buttons should perform the same action: saving the data on the page and moving to the next page. Users will be confused if the button sometimes moves them to a new page, but not always.

Standards apply to all aspects of the user interface: page and form layouts; color palates; headers and labels; placement and terms used on tabs, sub-tabs, buttons, and links; table layouts; calendars; keyboard shortcuts; abbreviations; etc.

User interfaces present a consistent set of conventions to users when the design team has a style guide of conventions to follow. In addition to following a guide, there needs to be style guide review for all screens. The review’s purpose is to verify that the page is consistent with the guide. Violations are noted and assignments are made for fixing any violations.

Examples of good practices:

1. Match the descriptions of titles and headers with the links that point to them.
2. Use a familiar page layout. For example: Put navigation on the left, links to special site tools such as “Preferences” on the upper right, and put featured items on the right side.
3. Only use attention-getting techniques for exceptional conditions or for time-dependent information.
4. Be consistent in naming and grammatical style for menu options, both within each menu and across the application.
5. Follow industry or company standards for buttons and links, for example:

Correct Term	Component	Usage	Incorrect Term for this Usage
Confirming Changes			
Apply	Action/Navigation button	Confirms changes made in current page without navigating to another page.	Go, Set, Submit, Update, Save
Save	Action button	Saves changes without navigating to another page, so that the user can continue modifying page contents.	Apply, Go, Set, Submit, Update

5. Error prevention and correction:

Does the application prevent error situations from occurring in the first place, and provide users with meaningful error messages?

Design Perspective:

One of the important principles of user-centered design is to prevent errors from happening or, when that may not be possible, to catch them immediately. There are a number of good practices that are known to reduce errors

- Whenever possible, have the application fill in data so that the user doesn't have to type it. This practice reduces the number of typographical errors.
- Use "Favorites" and "Recent Activity" lists to simplify navigation.
- Make the format of data-entry fields clear, especially dates, phone numbers, partial numbers, etc.
- Automatically validate data-entry values. When values are invalid, tell the user and indicate the correct range or format.
- Warn users about data value actions that could cause problems later.

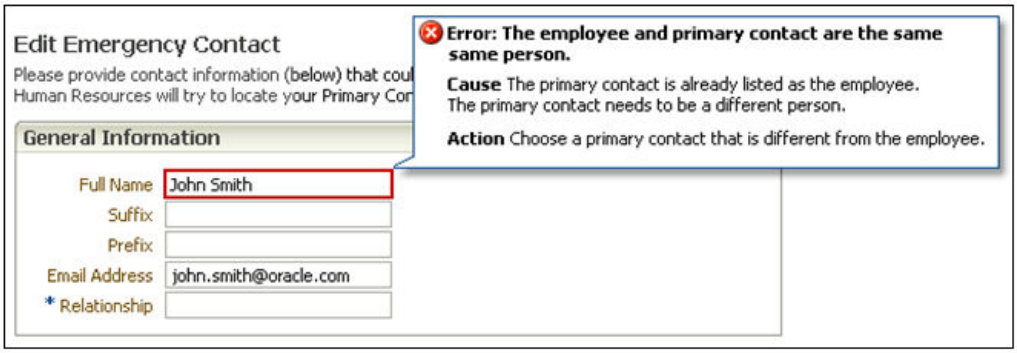
Examples of good practices:

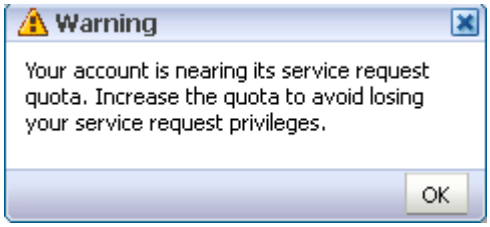
1. Reduce input errors by automatically filling, or auto-filling, fields that require repetitious data.

2. Make obvious validation checks. Examples of validation checks:

Range check	This checks that the data lies within a specified range of values. Example: The month of a person's date of birth should lie between 1 and 12.
Presence check	This checks that important data is actually there. Example: Users may be required to enter telephone numbers.
Type check	This checks that data is of the right type. Example: A number, text, etc.
Length check	This checks that fields have the correct number of characters. Example: A bank account number must be 10 digits.

3. Place function keys that can cause serious consequences in hard-to-reach positions.
4. Provide warnings about data that could cause a problem later.





6. Recognition rather than memory overload:

Does the application make users remember information, causing memory overload, or does the application make options visible?

Design Perspective:

People are notoriously poor at recalling detailed information, but are remarkably good at recognizing it. Our limited ability for recall makes using software applications particularly difficult. Users see one page at a time and must build up a mental model of the software. Many of the good practices described in the other heuristics in this document work because they reduce the need for recall; for example, consistency in the use of terminology and navigation provide feedback about where the user is, laying out screens so that similar items are grouped and auto-filling entries that the users has previously entered elsewhere help fill in memory gaps.

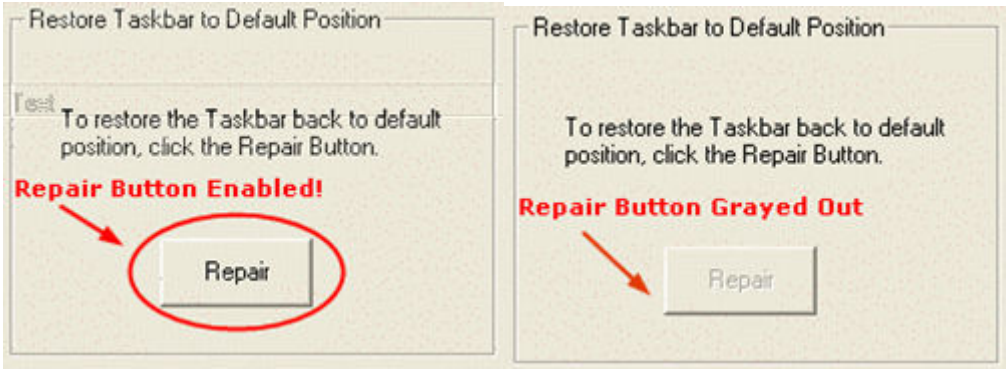
The pull-down list is a good example of a widget that allows users to select by recognition rather than type an option from memory. Better still is the pull-down menu in which frequently used items gravitate to the top of the list over time.

One frustrating aspect of many applications is the need to perform searches. In most cases, users must type an entry from memory and click on a search button or icon. This process induces many errors. People are not good at recalling details such as the correct spelling or whether it's better to search by last or first name. When the search function returns an empty list, or a huge one, users have to rethink their strategy. There are several ways to design a search function that reduces errors. The best strategy is to reduce the need for searching. A good practice is to display a list, perhaps of favorite items that users have searched for in the past, and allow users to choose from it.

Finally, as mentioned before, retaining a user's work on one page keeps it in context of the task. Using pop-up dialogs and mouse-over tool tips avoids sending users to a separate page, where they may forget that they were doing.

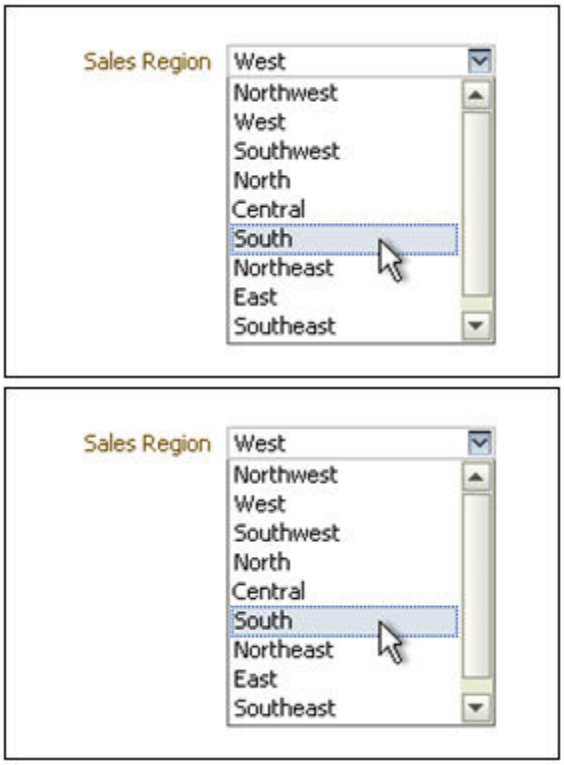
Examples of good practices:

1. Make labels and controls of currently inactive functions gray.



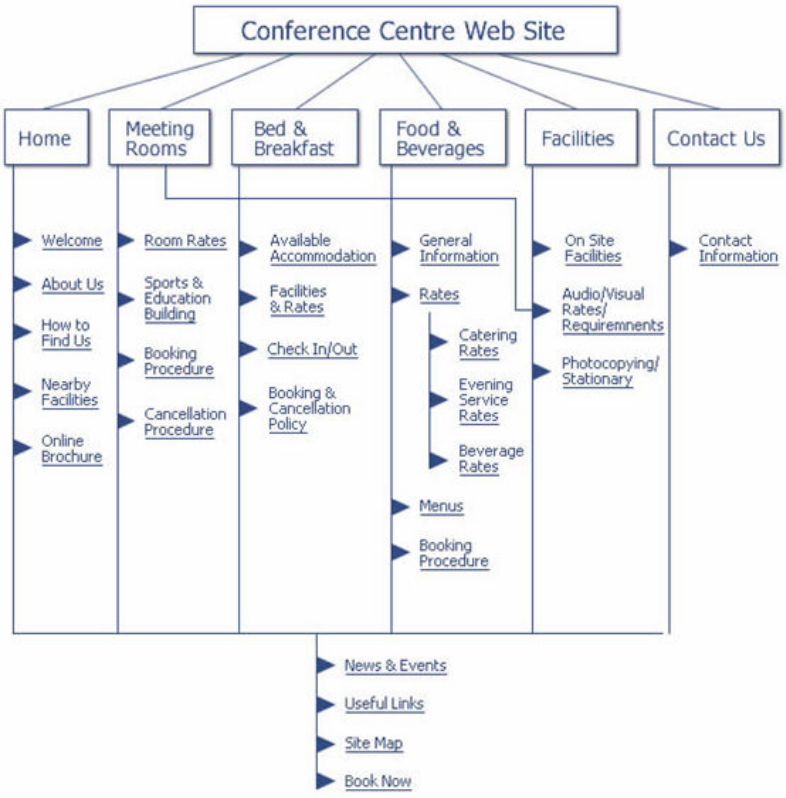
2. Use borders to separate meaningful groupings.

3. Whenever possible, allow users to choose from a list of options rather than type an entry.



4. Use mouse-over tool tips to explain functions that are used infrequently.

5. For complex applications, provide a site map that users can access easily.



7. Efficient to use:

Does the application allow both novice and expert users to get their work done quickly and efficiently? The application should allow users to tailor frequent actions.

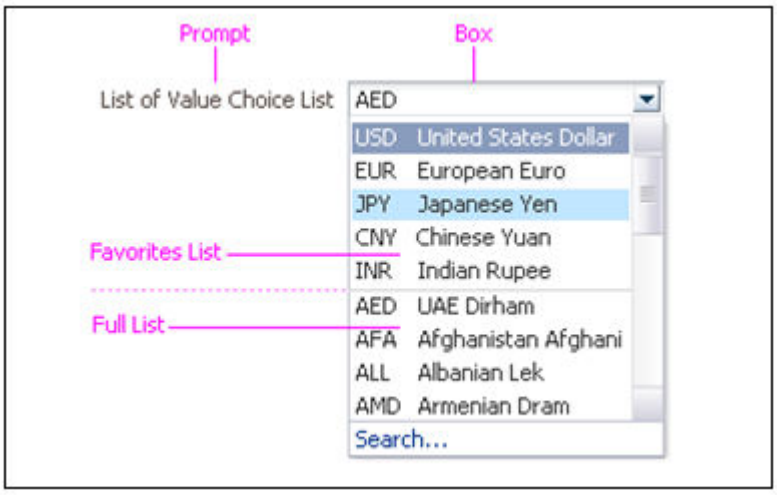
Design Perspective:

Most of the focus on usability is on first-time or early use of applications, which is important because that strategy can provide barriers to later use. Over time, users want to be able to take advantage of their experience. For example, if they know the format of dates, they may want to just type the date rather than using a date picker. Function key accelerators help experienced users without hindering new users as to shortcuts put into pop-up menus accessed by a right-click on the mouse.

The use of the Shift key for contiguous selection of items and the CTRL key for non-contiguous selection should be enabled on tables and lists. Type-ahead capability allows users to keep working when the application is not quite ready for input. Recent actions and "Favorites" lists often help experienced users more than new ones.

Examples of good practices:

1. Use accelerators, unseen by the novice user, to speed up interactions for the expert.
2. Make frequently-used items the first choice in lists.



3. Provide "Bookmarks" or "Favorites" lists for frequently used choices.
4. Allow type-ahead whenever possible.
5. Auto-fill fields whenever possible.

8. Simplicity and appeal:

Does the application make tasks simple and appealing? The application should streamline information because every extra unit of information in a dialogue page competes with the relevant units of information and diminishes their relative visibility.

Design Perspective:

Effective graphic design means more than creating pages that are pleasing to the eye. Good design makes the organization of a visual page clear at a glance and draws users' attention to what it most important on the page. There are several key components that contribute to good graphic design. One of the most important is the use of color. The core Oracle applications were designed to follow a color palate. That palate purposely uses muted colors that don't overwhelm the look of the page. In addition, the palate was designed to make sure that adjacent fields or areas do not have widely varying hues that can confuse the eye. In configuring or customizing an application, the design of new pages or windows should not deviate from that palate.

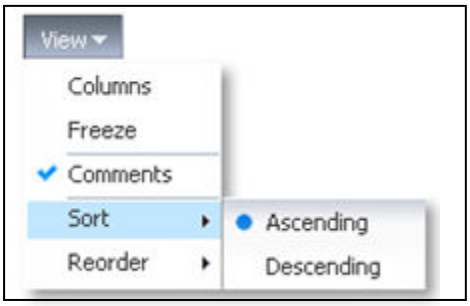
Typically, special highlighting beyond what is used on core application pages should not be needed. Occasionally, underlining a key word in instruction text may be used to call attention to it. But highlighting beyond what the core application pages use should be avoided.

One of the tools that can simplify a design is the use of icons, which are heavily used in modern enterprise software. Use the same icons that the core application uses whenever possible. Creating new icons for a customization is a challenge. It is difficult to create effective icons for new functions. Effective icons are recognizable, that is, more than two-thirds of users are able to identify the function the first time. One factor that helps recognition is the use of a visual representation that maps to the function. For example, this icons looks like a calculator:

If clicking on this icon brought up a calculator, the connection might be clear. But if it were used to calculate a total, its meaning might be ambiguous. The only way to know if an icon is recognized by users is to test it with them in the context of a task they perform. If such testing is not feasible, it is best to avoid using a new icon when designing a customization.

Examples of good practices:

1. Make sure your content is written for the application and not just repackaged from a paper source. Reading from a screen is different from reading from a paper document.
2. Make icons visually distinct and ensure that users understand the link between the icon and the concept it represents. Test new icons with users for recognizability.
3. Use progressive disclosure, that is, reveal information to the user in small, manageable amounts through the use of Show Details buttons, pop-up boxes, pull-down menus, and hierarchical menus.

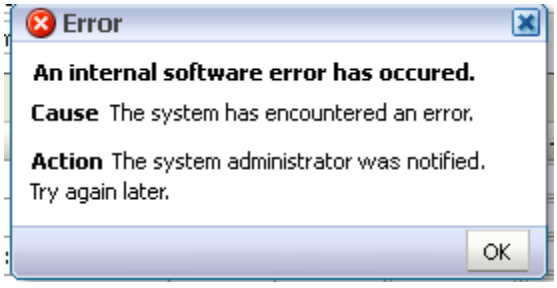


9. Be tolerant and reduce cost of errors:

Does the application allow users to recover and move forward in achieving their goals without unnecessary frustration? Error messages should be expressed in plain language, indicate the problem precisely, and, if possible, suggest a solution.

Design Perspective:

Poorly designed error and warning messages have plagued software applications since they first appeared. Poor messages vary from incomprehensible to insulting. Effective error and warning messages tell the user what happened and suggest solutions:



Writing them requires understanding the context in which they appear. One of the challenges for designers who are customizing an application is managing these messages. Some error and warning messages will come from the core application. When they need to be made more comprehensible, the application designer may be able to modify them or intercept them. If not, some other means for helping the user may be needed, such as a tool tip. When an application is tailored extensively, new messages may be needed. Ideally, the implementation team has access to a technical communicator with experience at writing error messages. If not, then it is the designers' responsibility to write them.

All error messages should provide some basic information:

- What is the problem?
- What, if anything, can the user do to fix it or recover from it?
- What, if anything, can the user do to prevent this problem in the future?

Consider the content each message needs. The main goal of error messages is to tell users what's wrong and then get them back to their task as quickly as possible. Give serious thought to how much detail your users need about each problem. Some might benefit from seeing specific information on a problem, such as a server timeout, but weigh the needs of those users against the needs of people who just want to know two things: What's the problem? What can I do to fix it?

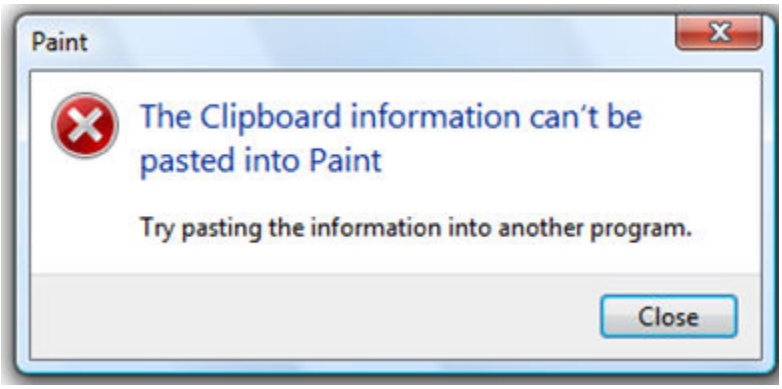
The tone of the message is important. Never blame the user. You can easily diffuse blame in your error messages

by simply stating what the user needs to do. For example, an error message informing the user that the password he or she typed is too short might say: "Type a password that is at least five characters long" instead of "You typed a password that is too short."

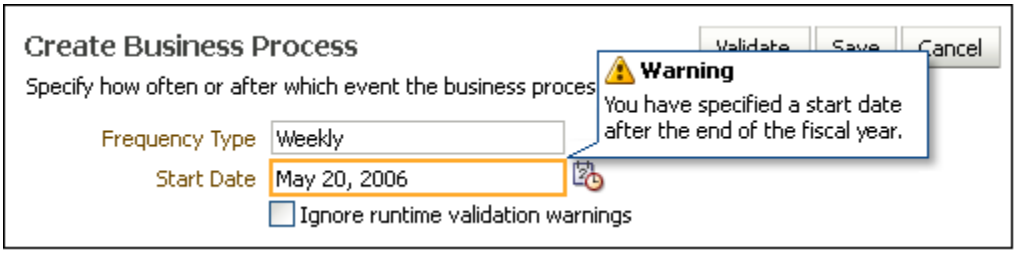
Finally, the format of the messages you create should be consistent with the format of messages from the core application. It is best to show the user a consistent format than to create your own. For more on user assistance messages, see the UX Direct: Messages collateral.

Examples of good practices:

1. Write error messages constructively, avoiding overt or implied criticism of the user.



2. Suggest a solution whenever possible: "Did you mean *Web site* when you typed *web stie*?"
3. When errors are not easily diagnosed, make suggestions for a solution based on probability: "Incorrect password. Check to see if your caps lock key is on."
4. Make error messages consistent in look and behavior.
5. Make error messages visible and highly noticeable, both in terms of the message itself and how it indicates which dialogue element users must repair.



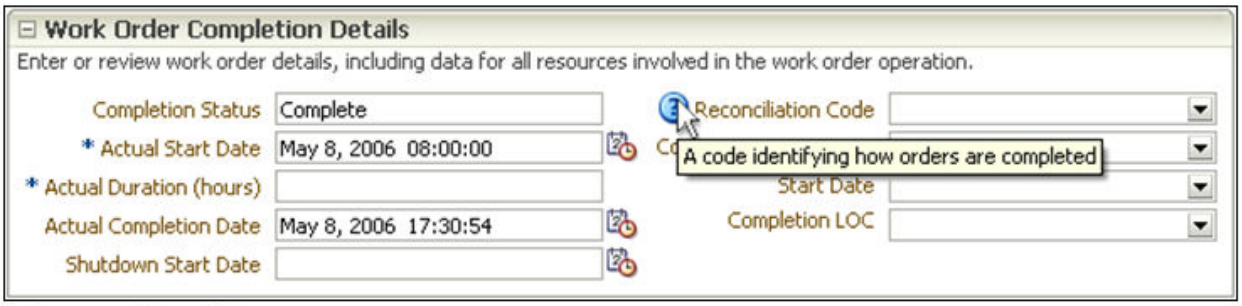
10. Help support:

Does the application provide clear instructions on how to perform steps when users are struggling to complete their tasks? Simple and concise instructions, prompts, and cues should be embedded in the application.

Design Perspective:

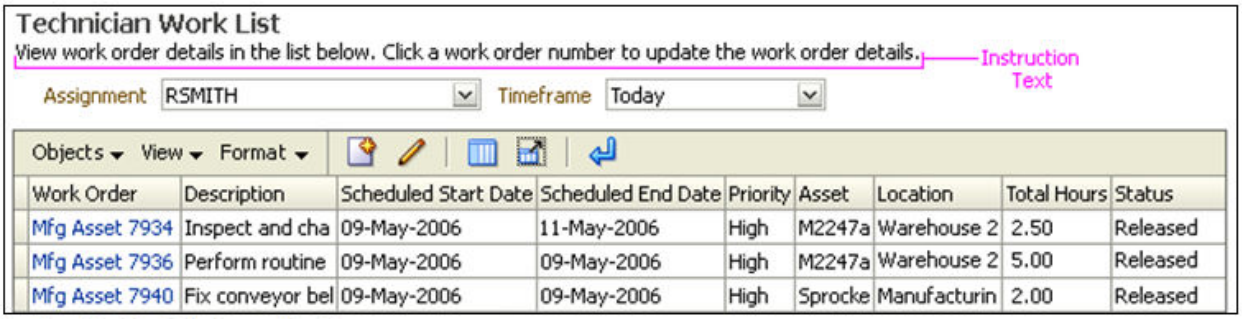
Online or embedded help takes several forms. For Oracle products, the forms include:

- **Help Icon**: Can be configured to link to a Help system, to display a brief description in a tooltip, or both.



Help Icon with Tooltip

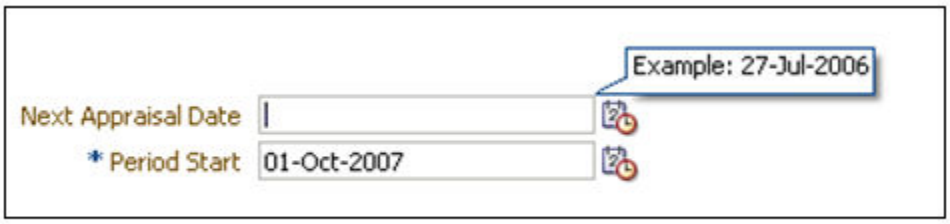
- **Instruction Text**: Static text or a note window with directions or guidelines for performing a task.



Work Order	Description	Scheduled Start Date	Scheduled End Date	Priority	Asset	Location	Total Hours	Status
Mfg Asset 7934	Inspect and cha	09-May-2006	11-May-2006	High	M2247a	Warehouse 2	2.50	Released
Mfg Asset 7936	Perform routine	09-May-2006	09-May-2006	High	M2247a	Warehouse 2	5.00	Released
Mfg Asset 7940	Fix conveyor bel	09-May-2006	09-May-2006	High	Sprocke	Manufacturin	2.00	Released

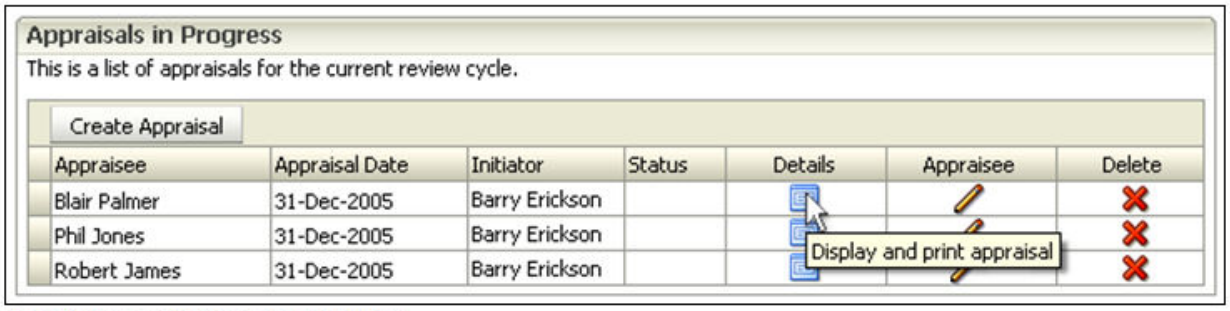
Instruction Text Below a Header

- **Hint Text**: Tooltip with field formatting and validation rules. It helps avoid user error by displaying rules for fields that accept direct user input.



Hint Text for an Input Field

- **Description Text**: Tooltip with a brief explanation of a UI element. It is especially useful for elements that do not support other forms of help, such as icons, images, and list options.



Description Text for an Iconic Button

The following guidance is taken from Oracle's Browser Look and Feel Guidelines.

Effective help design depends on the same information as does successful application design — detailed knowledge of the application's users. Some applications are used by a single type of user. However, most applications are used by different groups of users with differing software skills, responsibilities, and domain knowledge. Differing groups of users may spend their time working in different parts of an application; thus, help must be tailored to meet the audience needs for each part of the application.

Here are several questions that help determine which groups need help in which parts of the application.

- Do different groups of users typically use different tabs or groups of pages in the application?
- Which high-frequency tasks do each group of users find difficult to complete successfully?
- On which pages do users perform these tasks?
- How great is the difficulty? Can they figure out how to do it by themselves? If so, do they remember how the next time they perform the task?

Web applications have multiple methods of providing help, each with a different scope (component, section, page, or group of pages). This information should be gathered before developing help for the page whenever possible.

- Is the problem caused by a single component, or by the interaction of a group of components?
- Is the problem caused by a shortcoming in UI design that cannot be fixed for some reason, or by a lack of domain knowledge (not understanding technical terms, or not knowing which choice to make)?
- Will a brief set of instructions suffice to address the issue?

It is recommended to first identify individual fields requiring help, and then determine whether more general information is required:

- Provide tool tip Description Text for every iconic button.

- If a field prompt or button label is not self-explanatory, and this cannot reasonably be fixed by editing the prompt or label, provide description text with a fuller explanation.
- If an editable field has formatting or validation constraints, use Hint Text. If additional directions or related information are also needed, use Instruction Text.
- If a read-only text element, such as a column header, is not self-explanatory, use a Help Icon with definition text and a link to the Help system for further information if needed.
- If a list item requires additional explanation, use description text.
- When users need directions to complete a task that involves multiple components, use Static Instruction Text. If additional information is needed, provide an icon to link to the application's Help system.
- If an overview of a multi-page task or business requirements is needed, provide static instruction text with a help icon to link to an overview topic in the Help system.

Examples of good practices:

1. Use tool tips that provide useful information instead of repeating what is already on the page.
2. Whenever possible, Help should be context-sensitive, such as providing a link directly into the relevant section of Help. For example: “See Enabling International Support.”
3. Allow the user to switch easily between Help and their work.
4. Provide several access methods to the Help text: table of contents, index, search and embedded hyperlinks.

Appendix B: Heuristic Evaluation

Evaluator's Form

Name: _____

Date: _____

Describe the usability problem:

_____

_____

_____

_____

What task and step was being attempted when the problem occurred?

_____

_____

_____

_____

Where in the product did the problem arise?

_____

_____

_____

What solution will solve the problem?

_____

_____

_____

_____

ORACLE

Oracle Corporation

Worldwide Headquarters
500 Oracle Parkway
Redwood Shores, CA
94065
U.S.A.

Worldwide Inquiries
Phone
+1.650.506.7000
+1.800.ORACLE1

Fax
+1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment