Client-side Web Design - Designing our app

A brief consideration of useful hints and tips for designing an application.

Contents

Considerations

As we design a product's interface, we normally have to think our way through the following salient points and considerations:

Part 1

- tasks and activities a user can and should be able to perform with the product
    - ie: what is the considered scope of the product's functionality?
- as we consider each task, how will the interaction develop and be processed?
    - in effect, what are the expected steps and actions for the user and the product?
- we need to consider carefully the overall visual style or appearance of the application
    - eg: visual design and layout for the basic page templates or screen layout - fonts, colours, typography and iconography, any branding...
- what are the defined **places** in our application?
    - eg: pages for a website, navigation controllers and panels for mobile apps, levels in games, and so on...
- how does our user actually navigate between these **places** within our application?
- as we consider further our app's places, what content and layout will be

presented to the user in each *place*.
    - which controls are available, how will they be presented, arranged, and so on?

Part 2

- how will the user interact with these controls?
    - ie: just mouse and keyboard, is touch accepted?
    - are there behaviours associated with these controls?
- are there any events within our application that are not triggered by the user?
    - eg: timer driven events, remote calls and services, backup protocols, automatic updates...
    - are any behaviours actioned during such events?
- does the application store, request, manage any data?
    - what type of data, where, format, protocols, services...
    - how do we present this data on-screen and to the user?
- is there a naming scheme for interface and interaction elements?
    - eg: data, elements, places, objects, controls, navigation, and any other pertinent concepts...

Part 3

- error handling scheme for the app
    - how will the user be informed? will the user have the option to gracefully recover from errors etc?
        - are there defined user roles in the app?
    - what actions, privileges are permitted per role?
        - how do our users request or find assistance within the app?
    - is it an active system or passive? ie: interactive or reference based documentation, tutorials, videos, discussion forums etc...
        - how is the app structured to promote app guidance for users through tasks?
    - help for the users to work out how the app actually works...

Part 4

To be able to fully consider and plan for each of these outlined points, we will need to engage in a number of related tasks. For example, gathering requirements and their analysis.

We will also need to understand our user base, effectively the target audience for our application. This will include their characteristics, their requirements, and how they

intend to interact with the application. Therefore, as designers and developers we will need to understand the type of work they want to complete, the inherent tasks, and the effective problem domain.

To a lesser degree, this will also require an understanding of the technology requirements. Such technology choices, for example chosen languages etc, will have an impact upon how and what we are able to design and provision for our users.

We will need to consider prototypes, mockups, design documentation and specifications and, of course, testing.

Principles for usability - intro

Let us return to usability, and now consider some of the underlying design principles that help guide our considered designs.

For example, we'll now consider some of Don Norman's design principles for usability. These were detailed in his book, **The Design of Everyday Things**, first published in 1988. In this book, Norman introduced a set of basic design principles and concepts that are now considered foundational in the general study of usability. These are as follows,

- consistency
- visibility
- affordance
- mapping
- feedback
- constraints

We will now work our way through these design principles in greater detail.

Principles for usability - Consistency

- one of the primary ways our users learn is by discovering *patterns*. When presented with new situations, they become more manageable and easier to navigate once referenced to an existing pattern of knowledge. This can then be applied some way towards the new situation. *Consistency* is key in helping our users recognise and apply such patterns.
- Generally speaking, things that look the same should perform the same general way. For example, if a user sees a particular button in a given colour, they expect, quite rightly as well, that a similar pattern will follow for other examples of that button and colour.

- Behaviour and actions should also follow a similar pattern. If there is a sound, animation, vibration etc for a form action, then that should be applied for other forms and similar actions. We need to create and promote these patterns for our users.
- Inconsistency in design and application can cause confusion and increase potential cognitive overload for our users. Forcing users to memorise exceptions may also increase the chances of users becoming resentful of our application, which naturally increases the chances of them finding a different application.
- internal application consistency is naturally crucial for our users, but we also need to consider external consistency. For example, if we develop an application for a specific OS, whether it be mobile or desktop, we should try to conform to UI guidelines provided for that OS. Again, consistency with these platform guidelines helps lessen the number of concepts and patterns a user has to learn.

Principles for usability - Visibility

- users normally learn what functions may be performed within an application by visually inspecting the UI and, effectively, determining what controls are available.
    - eg: the available menus, menu items, icons, buttons, links, tools etc...
- for stepped, sequential tasks, it helps the users to understand and see the *next* step if the controls are clearly defined and labelled, and appear in a consistent and obvious location.
- according to this principle, usability and learnability are naturally improved for a user when they are able to easily see what commands and options are available to them. Therefore, controls should be clearly visible, contextually appropriate (ie: don't include buttons etc that are not relevant to the given task at hand...), and positioned where a user would expect to find them...
- functionality within an application that is not visually represented can be hard for a user to discover. For example, keyboard shortcuts will often save a user time and effort, in particular *expert* or *power* users, but this will often be a poor choice as the only way to complete a given command etc...This is one of the reasons why we often see shortcut commands listed next to their UI option in menus etc. They become easier to view, associate, and learn as they appear in an obvious, intuitive visual manner.

This concept of visibility does not, necessarily, infer that all options and possible functions should be graphically represented. For many complex applications, this would naturally be impractical, and the design would be overloaded with buttons, links, menus etc. Therefore, this is where careful and considered design is of vital importance. It becomes a balancing act between functionality and practical design. It

is also a good example of the value of context, and the application of visual *perspectives* relative to the current task and activity. Adobe's Photoshop is a good example of this application of design.

Principles for usability - Affordance

- this is a visual attribute or physical property of a given object or control. It gives the user clues to the operation or functionality of an object or control. In effect, affordances are the parts of the system that can be manipulated to allow a user to interact with the given system. A standard way of describing this is to consider door handles. For example,
    - the door's doorknob invites a user to turn it to open the door. We say that the doorknob **affords** (basically, it allows) turning.
    - there might be a kick plate on the bottom portion of the door, which invites us to push the door as well. Again, it **affords** pushing...
    - A grab handle on a door invites a user to naturally pull the door open towards them. Therefore, this handle **affords** pulling. However, if the door swings open both ways, it may also afford pushing.

What we can see from this example is that the shape of the handles, and the nature of the door itself, present clues to the door's functionality and how it may be manipulated by a user.

If our door appears to invite a pull, and then requires a push, it will frustrate and annoy the user. We can, of course, apply this design concept quite easily to our UI designs and interaction patterns. We can use visual clues to make our controls look clickable or touchable. One common technique used to be the application of 3D, raised bezels to create the impression of buttons to click. A control or UI element may also give a user a clue to suggest that it should be clicked or used at a given point in an application process. For example, a slight highlight for a *submit* button as a user completes a form. Again, we could add a slight highlighting or change the colour of the search button as a user starts to type a query in a search box, and so on. We should also be mindful, as noted earlier, of OS conventions for such UI controls and elements.

Design conventions have been developed for a reason, and they offer a useful reminder of how patterns can easily be developed relative to an interface. For example, it is not inherently apparent that text underlined with blue is a link on a webpage, but we have become accustomed and programmed to accept this simple fact as a given for such text.

Another example in many applications is to change the shape of an item relative to its available action or option. For example, if we again consider webpages we often see a mouse cursor change shape relative to a given link type. It may change from a hand to an arrow to an hour glass, and so on.

Principles for usability - Mapping

If a user clicks a button or activates a control such as a sub-menu, there is, normally, a relationship between the performed action and the expected result. In effect, this is mapping between a given control and its behavioural effect.

As with interaction in general, such mappings should be logical, explicit, and as straightforward as possible. We can generally achieve this result by using descriptive labels or icons on buttons and menus, and by using consistency, as outlined earlier.

Again, controls should be positioned in a logical manner, adhering to conventions where possible. For example, there are many conventions in both UI design and interaction, and the real world, that help guide our design decisions. A slider bar normally represents sliding up to increase and down to decrease. Sounds obvious, but change it or modify it in some other sense and our user is thrown out of an agreed convention. If you do need to change conventions, for whatever reason, you will also need to ensure this is strongly reinforced and tested in the application's training and learning material.

Principles for usability - Feedback

Feedback plays a crucial role in reinforcing users' perception, expectations, and general experience when using an application. If a user presses a button, and there is a delay, then the user will need to know whether it is necessary to press the button again, wait, try another option and so on.

The principle of feedback states that designers should offer users confirmation or a simple acknowledgement of the result of an action. Good or bad, successful or unsuccessful, a user should be given feedback of the result. We are also able to distinguish two general types of feedback,

- **activational feedback**
    - provides evidence that a given control was actioned successfully.
    - eg: a button pressed, menu item selected, slider control moved to a new position
    - feedback may be offered visually, in a tactile manner for physical controls, with an audible alert, and so on...

- **behavioural feedback**
    - provides evidence an action etc has had an effect of the application, system...
    - eg: an application may close an open, active window after a completed successful action, a dialog window and status message may be shown, audible sound, and so on...

Principles for usability - Constraints

- apps and interfaces need to be designed and tested to prevent invalid states
    - incorrect, invalid user interaction, invalid actions...
- constraints may take various forms
    - check correct relationships between elements and actions
    - check elements active only as needed
    - actions only performed when default data etc available
    - menu items active relative to contextual requirements
    - physical products often display such constraints

Principles for usability - Naming

Another consideration

- names and labels key aspect of human communication, thought, understanding...
    - also an important consideration in design
- naming helps users understand the application
    - their current location relative to navigation
    - the data and information they are viewing
    - action they can and cannot perform...
- good naming helps a user form a correct mental model
- do not confuse naming with the use of technical jargon and terms
- precise, consistent naming helps us form unambiguous instructions, help, feedback...
- naming helps identify as well as differentiate between aspects of the design and functionality
- names should be unique relative to the context and the application
- namespaces are useful relative to application design and development

Principles for usability - Naming guidelines

A few guidelines and thoughts on naming...

- does the name accurately reflect and describe its intended target?

    - consider the action of the element relative to the name
- is the name clear, concise, and free of ambiguity?
- use concise, easy to remember names
    - better than longer, hard to remember descriptions
- does the name inherently assume prior knowledge from the user?
    - consider naming relative to perceived domain knowledge
- acronyms are useful, but assume prior knowledge of the domain
    - be careful when using acronyms, and consider cultural bias
        - eg: VAT well known in Europe
- carefully consider capitalisation, and ensure consistency for chosen pattern
    - eg: This Is Capitalised...This is Capitalised...This is not Capitalised (fully)...
- users should be able to pronounce a name...not helpful if they have to check first

References

- Norman, D. *The Design of Everyday Things.* Basic Books. 2013.