[L6P2]

From the Magician's Hat: Designing the Product

Product design

Product design is the design of a software solution that meets the requirements identified earlier. Often, programmers are required to just implement a given specification. However, at times, programmers are directly involved in product design. Even if programmers do not explicitly partake in 'product design', their 'products' are still used by others. As an example, a module written by one programmer may be used by another module written by a separate programmer. It is therefore important to spend effort in designing the product to solve the users' problem in the most optimal manner, not just the way that is 'cool', 'interesting', or 'convenient' to the designer. Below are some guidelines (adapted from [1]) that may be applicable when you design your product.

Have a vision

It is helpful to have a grand vision for your product. Rather than 'just another *foo*', you can go for something like 'the most streamlined *foo*', 'the most versatile *foo*', or 'the best *foo* on the platform *bar*'.

Put vision before features

Combining a multitude of good features does not automatically render a good product, just as duct-taping a good camera to a good phone does not result in a good camera phone. Resist the temptation to make up a product by combining feature proposals from all team members. The vision for your product should be the foremost consideration as it dictates the features for your product. Defining a vision having decided on the features is like shooting the arrow before deciding where the target should be. It is very important that you are clear about the vision, scope and the target market for the product. These should be documented and made available to others in the team, as well as any other stake holders.

Focus features

Given the tight resource constraints of most projects, do not be distracted by any feature that is not central to your product's vision. Ditch features that do not match the product vision, no matter how 'novel' or 'cool' they are.

Do not provide features because you think they 'can be provided at **zero cost**'. There is no such thing. Every feature comes with a cost. Include a feature only if there is a better justification than simply because "it costs nothing".

Some teammates might try to squeeze in their '**pet features**' for a sense of ownership. Reject them gently but firmly. If 'password security' is not essential to your product vision, do not add it even if a team member claims that he has the code for it and can be included in 'no time'.

Aim in **one direction**. Avoid having multiple 'main' features that pull the product apart in different directions. Choose other 'minor' features so as to strengthen the 'main' feature. One 'fully baked' feature is worth more than many 'half-baked' ones.

Do not feel compelled to implement all '**commonplace features**' found in similar products before moving on to those exciting features that will set your product apart. A product can survive with some glaring omissions of common features if it has a strong unique feature. The

success of the iPhone is a prime example, despite missing on 'common' features such as 'SMS forwarding' when it was first released.

Focus users

Larger targets are easier to hit but harder to conquer. It is better to target your product for a well-identified small market (e.g. faculty members of your own university) than a vaguely-defined but supposedly larger market (e.g. all office workers). Taking this to the extreme, it is even acceptable to identify a single real user and cater for that person fully, rather than trying to cater for an unknown market comprising an unknown number of users.

It is critically important that you "see the users' perspective" when deciding on the features to include in a product. What is 'cool' to you may not be 'cool' to the users. Do not be tempted by the sense of accomplishment you get from implementing 'technically challenging' features when these do not provide significant benefit to the user.

Get early feedback

Get feedback from the potential users. Force upon yourself to use your software; if you do not like it, it is unlikely someone else will.

You can use the requirements specifications or an early prototype as the basis for soliciting feedback. Another good strategy is to write the user manual very early and use that to get feedback. Even better, you can do an early UI prototype/mockup and get feedback from potential users before implementing the whole system. UI prototypes are great in answering the question: "Are we building the right system?" A UI prototype need not have the final 'polished' look, and it need not be complete either. A mock-up can be drawn on paper, using PowerPoint, using drag-and-drop UI designers that come with IDEs such as Visual Studio, or using special prototyping tools such as Balsamiq.

Do not be afraid to change direction based on early feedback, since there is still time. This is precisely why you need to obtain feedback *early*.

Gently exceed expectations

Plan to deliver a little bit more than what is generally expected or previously promised. This will delight the user. Conversely, delivering even slightly less than promised creates a feeling of disappointment, even if what you managed to deliver is still good.

> *The design of the UI is important because even the best functionality will not compensate for a poor user interface, and people will not use the system unless they can successfully interact with it*
> *[adapted from the book Automated Defect Prevention: Best Practices in Software Management]*

Design the UI for the user

The UI is for the user; hence design it for the user. It should be easy to learn, understand, remember and navigate *for the user* (not you, the developer). A UI that makes the first-time user wondering "where to start" is not an intuitive UI.

> *...users aren't stupid; they just simply aren't programmers. Users are all smart and knowledgeable in their own way. They are doctors and lawyers and accountants, who are struggling to use the computer because programmers are too 'stupid' to understand law and medicine and accounting [http://tinyurl.com/stupidprogrammers].*

Usability is king

Do your best to improve the usability of the UI. Here are some things to consider:

- First, make it presentable. If your team lacks graphics expertise, it is a good idea to just stick to standard UI elements without trying anything fanciful. For example, this page shows fourteen different UIs developed by students for software performing generally similar tasks.
- Minimise work for users. Here are some examples:
    - Minimise clicks. If something can be done in one click, do not force the user to use more. Stay away from 'cool' but useless UI adornments such as unnecessary animations (note: when used appropriately, animations can convey additional information to the user. For example, when a user clicks the 'minimize' button, an animation can be used to inform users that the application is now hidden within the task bar).
    - Minimise pop-up windows. Some messages can be conveyed to the user without using a pop-up dialogue, which would otherwise require him to close the window.
    - Minimise unnecessary choices. Do not distract the user's thoughts with things not directly related to the task at hand.
- Minimise chances of user error. For example, place the 'delete permanently' button in an area that is unlikely to be clicked by mistake. If a user is not supposed to perform a certain action at a certain stage, then disable/remove that action from that stage. Seek confirmation with the user before executeing a user command that is irreversible.
- Minimize irreversible actions. It is preferable to have an action executed right away while providing an 'undo' feature, rather than annoying the user with frequent 'are you sure' dialogs.
- Wherever possible, provide an avenue to recover from mistakes and errors. Suggest possible remedial actions, in addition to displaying an understandable error message.
- Minimise frequent switching between mouse and keyboard input. Minimise 'travel distance' taken by the mouse by placing all related click items close to each other.
- Within the UI, use terms and metaphors familiar to users, and not developer jargon. Do not use different terms/icons to mean the same thing.
- Make common tasks easy, rare tasks possible. Refrain from engaging everyone with tasks that concerns only a few. For example, when most users are fine with a default install location, do not compel everyone to click 'Yes, install here'; instead, provide a choice to 'Change install location' that can be clicked if required.
- Make the UI consistent. It may be a good idea to let one person design the whole UI.
- The UI should not leave the user wondering about the current system status. For example, "is it still saving, or is it safe to close the application now?"
- Minimise things that the user has to remember. For example, "which save location did I choose in the previous screen?"
- A good UI should require very little user documentation, if any at all. To have it "explained in the user guide" is not an excuse to produce unintuitive UIs.
- Think as a user when designing the UI. Even naming matters. In one case, renaming a button increased revenue by $300 million! [http://tinyurl.com/millionbutton]

Be different, if you must

There is no need to follow what everyone one else does, but do not do things differently just to 'stand out'. Deviating from familiar interaction mechanisms might confuse the user. For example, most Windows users are familiar with the 'left-click to activate, right-click for context menu' way of interacting, and will be irritated if you use a different approach.

Name it well

Do not underestimate the benefits of a good product name. Here are some things to consider.

- The name should clearly convey what your product is, and if possible, what its strengths are.

- The name should make sense to your target market, not just you.
- If you have long-term plans for the product, check if there is already another product by that name. (Did you know that Java was first named 'Oak'? They had to rename it because of an existing little-known language by the same name.) You can even check if the name fits well into a web domain name and that the web domain name is still available. The '-' saved www.experts-exchange.com from having a totally inappropriate domain name; but you might not be so lucky.
- Be wary of using team member initials to form a product name. It would not mean anything to potential users. The product is the main thing; not you.
- Be wary of difficult-to-spell/hard-to-pronounce names, alterations of existing names, and names resulting from private jokes in your team; users may not find those amusing.
- Be aware that names starting with 'Z' will end up at the bottom of any alphabetical listing.

References

[1] *Practical Tips for Software-Intensive Student Projects* (3e), by Damith C. Rajapakse, Online at http://StudentProjectGuide.info