[Handout for L3P2]
From the Magician's Hat: Designing the Product

Product design

Product design is the designing of a software solution that meets the requirements identified earlier. Often, programmers just implement a given specification. But at times, programmers directly get involved in product design. Even if programmers are not called upon to do explicit 'product design', most things they produce are 'products' used by others (e.g. other members of the team). E.g. A module written by one programmer may be called by a module written by another programmer. Therefore, it is 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. Given next is some guidelines (adapted from [1]) you can consider applying during product design.

Have a vision

It is helpful to have a grand vision for your product. For example, 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 give you a good product, just as duct-taping a good camera to a good phone does not give you 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 is the first thing you should settle because it dictates the features for your product. Defining a vision after deciding features is like shooting the arrow first and painting the target where it landed. 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 and any other stake holders.

Focus features

Given the tight resource constraints of most projects, you cannot afford to 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 'cool' or 'useful' they are.

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

Some teammates might try to squeeze in their '**pet features**' for their own enjoyment. Reject them gently but firmly; if 'password security' is not essential to your product vision, do not add it even if one team member claims he already has the code for it and it can be added in 'no time'.

Aim in **one direction**. Avoid having multiple 'main' features that pull in different directions. Choose other 'minor' features in a way that they strengthen this 'main' feature even more. One 'fully baked' feature is worth more than many 'half-baked' ones.

Do not feel compelled to implement all '**commonplace features**' of similar products before you move to those exciting features that will differentiate your product. A product can survive with some glaring omissions of common features if it has a strong unique feature. Take note that the successful iPhone did not have many '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 own university) than a vaguely-defined but supposedly larger market (e.g. all office workers). To take this approach to an 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 of an unknown number of users.

It is critically important that you use the users' perspective when deciding what features to include in a product. What is 'cool' to you may not be 'cool' to users. Do not choose 'technically challenging' features if there is no significant benefit to the user.

Get early feedback

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

You can use 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 if 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 "are we building the right system?" question. Note that a UI prototype need not have the final 'polished' look, and it need not be complete either. It can be drawn on paper, done using PowerPoint, or mocked up 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, when there is still enough time left. That is precisely why you need to get 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 previously 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; 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 user wonder "where do I start?" the first time he looks at it is not an intuitive UI.

...user's 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 look presentable. If you do not have any graphics expertise in the team, just sticking to standard UI elements without trying anything fancy is a good idea. For example, this page shows different UIs developed by students for 14 software doing largely similar tasks.
- Minimise work for users.
    - Minimise clicks. If something can be done in one click, do not force the user to use two clicks. Stay away from 'cool' but useless UI adornments such as unnecessary animations (note: when used right, animations can be used to convey additional information to the user. For example, some operating systems use an animation when a user clicks the 'minimize' button, to inform users how the application can now be found in the task bar. As a result, users will never have to wonder, 'oops, where did it go?').
    - Minimise pop-up windows. Some messages can be conveyed to the user without requiring him to click a pop-up dialogue.
    - Minimise unnecessary choices. Do not distract the user by forcing him to think about things not directly related to the task at hand.
- Minimise chances of user error. For example, place the 'delete permanently' button in a place that is unlikely to be clicked by mistake. If a user is not supposed to do a certain action at a certain stage, simply disable/remove that action from that stage. Double confirm with the user before the software executes a user command that is irreversible.
- Minimize irreversible actions. It is better to obey the action right away but provide an 'undo' feature instead of annoying the user too often with 'are you sure' dialogs.
- Where possible, give an avenue to recover from mistakes and errors. In addition to an understandable error message, try to suggest possible remedial actions.
- Minimise having to switch between the mouse and the keyboard frequently. Minimise travel distance for the mouse by placing all related 'click locations' close to each other.
- The UI should use terms and metaphors familiar to users, not terms invented by developers. Do not use different terms/icons to mean the same thing.
- Make common things easy, rare things possible. Do not bother everyone with things that only a few will care about. For example, if you think most users will be OK with the default install location, do not force everyone to click 'Yes, install here'; instead, provide a button to 'Change install location' that is optional to click.
- Make the UI consistent. It may be a good idea to let one person design the whole UI.
- The UI should leave no room for the user to wonder about the current system status. For example, the user should not wonder "Is it still saving or is it safe to close the application now?"
- Minimise things the user has to remember. For example, the user should not wonder "Which save location did I choose in the previous screen?"

- A good UI should require very little user documentation, if any at all. On the other hand, "it is explained in the user guide" is not an excuse to produce unintuitive UIs.
- Think as a user when designing the UI. Even naming matters. Did you know that 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 be 'different'. 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 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 won't mean much to potential users. The product is the main thing; not you.
- Be wary of difficult-to-spell/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

---End of Document---