Notes - Design - Users and Mental Models

- Dr Nick Hayward

A brief intro to users and mental models relative to application and interface design.

Contents

Intro Mental models are formed as a user learns how to perform given tasks within an application. They are effectively creating a mental model of how your application works and what they need to do to operate it successfully.

Mental model A mental model is a conceptual representation in our user's mind of how a system works, and therefore how to operate the interface for this system. A user's mental model will naturally reflect their current understanding, and therefore this understanding is subject to change as they learn and gain experience using the application. It may also diminish or disappear as a user forgets details over time.

It is this mental model that a user will rely upon to reason appropriately for the given application and scenario. Users will also develop expectations of the application's behaviour based upon such mental models.

If we consider user's mental models, and then compare them to actual correct system behaviour, the system's implementation model, it can begin to explain many usability issues and problems. Our designs should aid a user to develop a correct mental model and accompanying habits.

Use with apps and UI How may we consider a mental model relative to our applications and user interfaces.

Well, such mental models will often consist of the following elements:

- interface appearance
- interface concepts, syntax, general rules…
- navigation map
- plans and strategies for accomplishing tasks and reacting to problems &c.
- heuristics, conventions…
- perception of application's implementation model

interface appearance As a user works their way through an application and its interface, they will form visual images of the *places* they have encountered and those that subsequently become familiar.

For example, various pages, screens, tabs, windows &c. within the application's interface. However, for most users these mental images will be vague and inherently imperfect unless, of course, the user has an eidetic memory.

Therefore, as a user becomes more familiar with an application's user interface, they become familiar with the general layout of the places they encounter frequently. The quality of these mental images will necessarily be affected by frequency of use.

For example, a general user will build up a mental image of an application, but this will often not include specifics such as menu layouts, the exact sequence of icons in a toolbar, or options within control panels, and so on. A user is unlikely to be able to sketch out in detail an application's interface from a mental model.

interface concepts, syntax, general rules... An application is designed to solve a problem or meet a specific requirement. Therefore, the syntax and rules used to solve such a problem are known as the *application domain*, the *business domain*, or the *problem domain*.

Dependent upon your system, the *problem domain* might actually be pretty small. The user of this application would, therefore, only need to understand a handful of concepts. More complicated or involved systems naturally demand greater, and more in-depth, knowledge and understanding of a domain.

Some applications, in particular the more complex and involved, will, therefore, have to be designed with the inherent assumption of experience and prior-knowledge. Effectively, a thorough understanding and awareness of the required domain gained via education, training, or experience. Hopefully, a combination of each of these options.

Other applications have to accept the responsibility of communicating and highlighting their domain's concepts to the user. Games, in particular role-playing or fantasy, are often seen as an extreme example of this concept. Their use of imaginary worlds, tools, and actions provide an extra layer to their given domain. Even more cursory, simpler games require the user to quickly learn their objects and goals, and therefore how they need to interact within the game's domain.

However, in many scenarios, users will often only require a cursory understanding of an application to begin usage. For example, users do not need to understand the inherent concept of URLs to be able to enter a website address in a web browser's address bar.

This is also true if the application is partly automated or simply follows a pre-defined path. Buying concert tickets online or ordering a package delivery are both examples of interfaces for applications where some degree of guidance, or hand-holding, occurs for the user.

Many complex applications, including common office suite software, still allow a user to get started quickly, for example with a simple text document. Many users will simply be unaware, or may not even care, about the myriad options available within such an application. This allows us to develop application learning based upon a user's initial, cursory understanding and usage of the given application.

navigation map As we consider our applications, we can see that many examples include the notion of places.

For example, places may include pages, screens, tabs, windows, and so on, which allow a user to visit given content and data.

As a user begins to differentiate between these places, and navigate to them repeatedly, they gradually form a mental *navigation map*. This *navigation map* effectively directs them back to different locations within our application.

Navigation will become a regular action required by users to complete a task within our applications. Repeatedly buy goods at an online e-commerce site, and you will build up a complementary navigation map for such sites. The interesting thing with this type of map is that users will often simply expect similar patterns at other e-commerce sites as well. Consider the number of such sites that follow the same basic pattern, and how many open source e-commerce tools and content management systems that also support similar design patterns.

The other interesting aspect of navigation maps is that there may be multiple ways to get to the same location within our application. Users may often not be aware of such multiple options, and those that are will often simply adhere to a preferred method and route. Consider the many options available within applications for printing.

A simple example of a website flowchart is as follows,

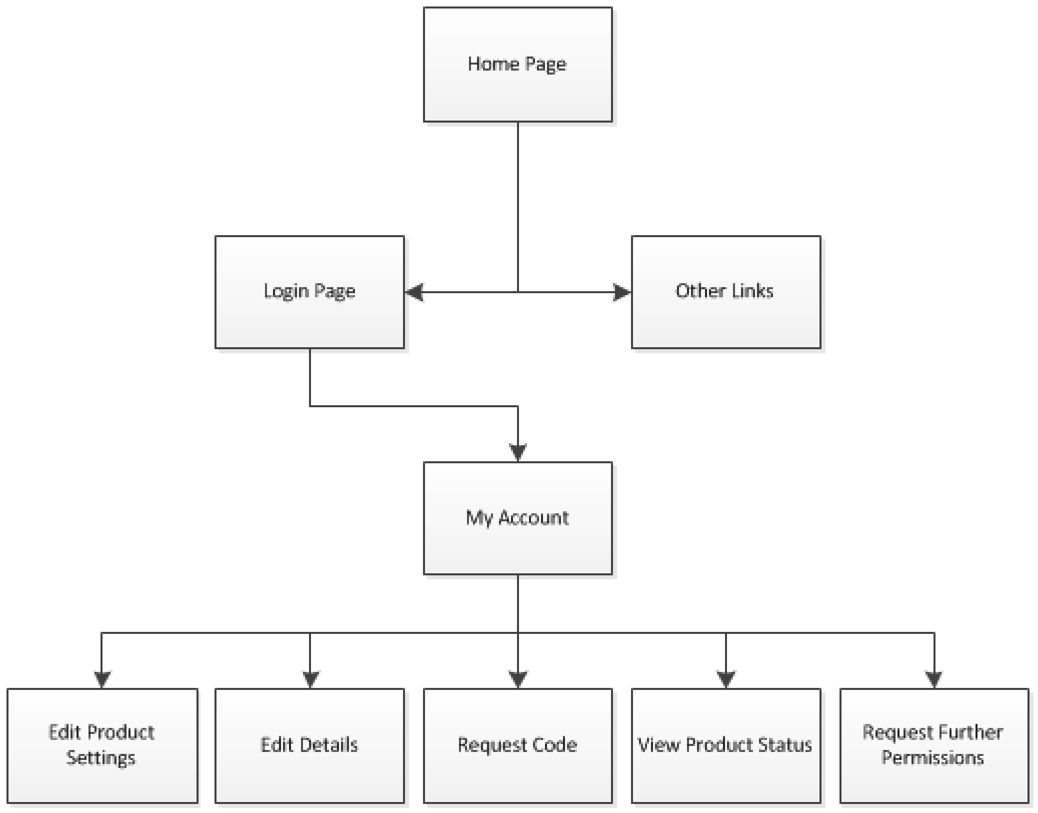

Figure 1: Simple example of website flowchart

We may then compare this example with a more detailed website,

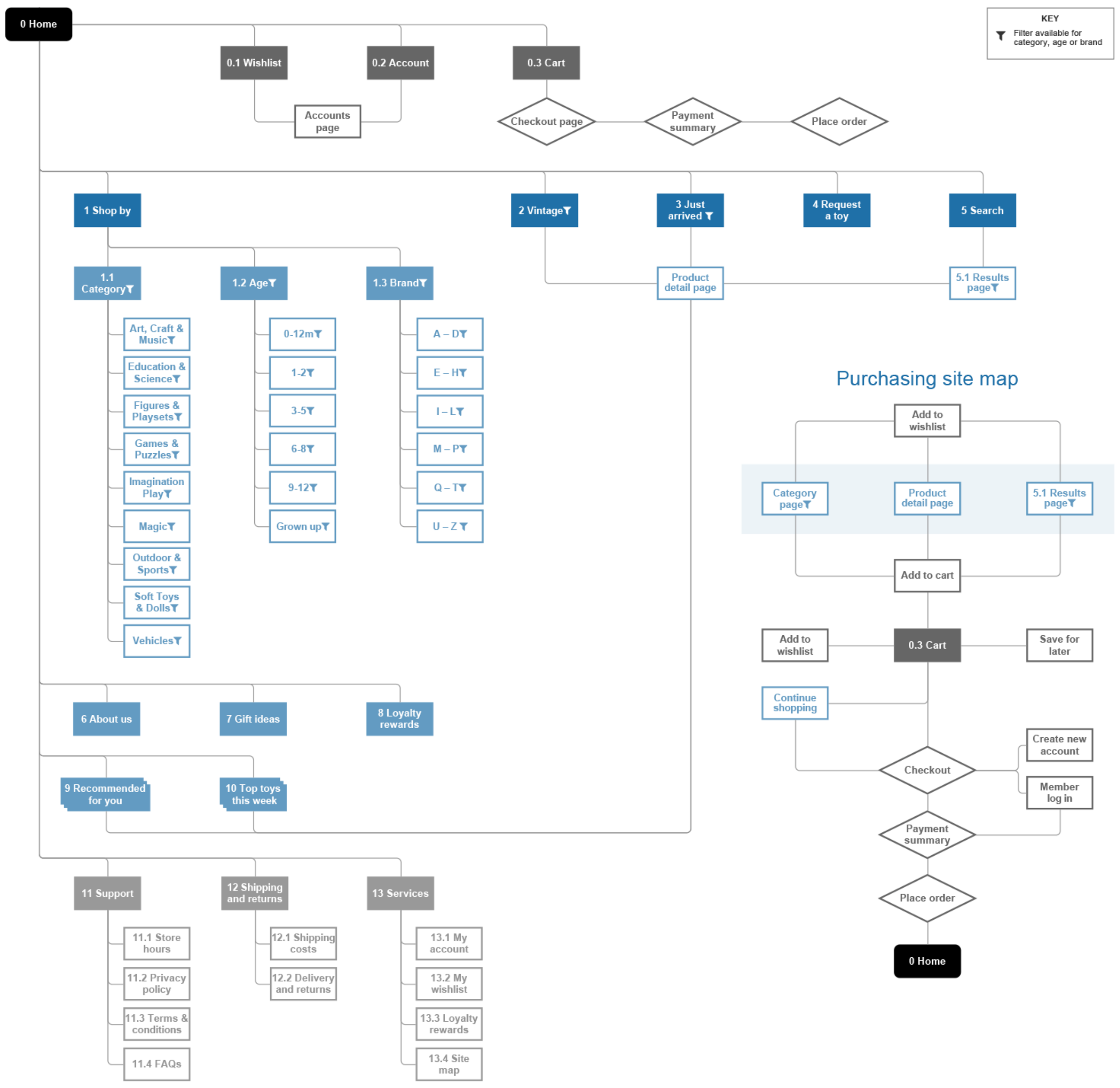

Figure 2: Detailed example of website flowchart

plans & strategies for accomplishing tasks & reacting to problems... Users will often memorise *plans of actions* required to achieve certain given tasks. An *action plan* might reflect a simple sequence of steps a user needs to follow. As a user becomes more experienced and comfortable with the application, they may internalise a required conceptual structure. This is often similar to a flowchart diagram which has various decision points and branches with steps the user may follow under different circumstances. It's worth noting that this mental depiction may not be complete, and may not always be correct. The mind has a habit of playing tricks on us.

A user may also not be explicitly aware of why a defined sequence of actions performs a given task, but they have memorised the sequence and are able to reproduce it as necessary.

A user taught by rote in a class or lesson may often fall in to this category. It will also often be the case if the user has stumbled upon a successful sequence by trial and error.

heuristics, conventions... The mental model developed and deployed by our user may also include general heuristics, or rule of thumb type guidelines, and other conventions picked up from a broader context. These may have been learned and added from experience, and subsequently applied to our system.

As an example, common UI elements may exist between disparate applications, or the application and operating system, thereby allowing a user to infer interaction patterns for an application.

perception of application's implementation model As a user becomes more familiar with an application, they often start to infer patterns for behaviour within the application. Whilst the code and implementation for an application should, more often than not, remain opaque to an end-user, this does not prevent a user from recognising such patterns from repeated general usage.

This is not actually a bad thing for most applications, as repeated patterns and manners can be used to our advantage as designers.

For example, a user may be able to create a new post within our application by selecting the button *Add post*, and then adding their content &c.. After saving their post, they notice that this new post appears at the top of the list with an associated link in the application's index. As they view other lists and indices, they should be able to infer that the most recent item or entry is at the top, and the first at the bottom. Bad design would arbitrarily change this pattern, and confuse the user of the application.

Communicating a mental model Mental models are not only useful for users but they are also part of the initial design process and thinking for us as interface designers. We will, naturally, have formed a conceptual mental model of our own for how the application should work.

The goal for us is to try to ensure that users develop a mental model that matches our designer's mental model.

One obvious way to achieve this goal is to provide a structured learning and educational infrastructure that clearly explains the product's concepts and operation. This may be output as documentation, training, demos, and so on. However, we also need to face the simple reality that many users will often not read the documentation or follow the tutorials.

Therefore, many users will still rely upon trial and error, simply starting to use an application and learning bits and pieces as they go along.

The visual presentation of an application's user interface provides cues and guidance to users on how to complete actions and tasks. The behaviour of the application provides feedback to the user as to whether those actions and tasks have been successful or not. Therefore, it is hoped that as a user develops familiarity with an application's user interface, their mental model will more closely approximate the designers.

Don Norman refers to these conceptual models as the *design model* and the *user's model.* The visual presentation and behaviour exhibited by the product's user interface is called the *system image.*

Therefore, to design a usable and approachable, in effect learnable, product, as designers one of our primary challenges is ensuring that the design model and system image are aligned so that the system image accurately portrays the design model. This should then enable our users to develop their *user model* as a close approximation of the design model.

Resources

- Card, S.K., Moran, T.P. and Newell, A. *The psychology of human-computer interaction.* Lawrence Erlbaum Associates. 1983.
- Krug, S. *Don't make me think, revisited: A common sense approach to web usability.* 3rd Edition. New Riders. 2014.
- Norman, D. *The Design of Everyday Things.* Basic Books. 2013.