

Designing for appropriation

extract of draft for new edition of HCI book
Alan Dix, www.hcibook.com/alan

In ethnographies of work settings one of the frequent observations is the way that people do not 'play to the rules' they adapt and adopt the technology around them in ways the designers never envisaged. Think to your own experience perhaps you have used a screwdriver to open a tin of paint, or a heavy text book to prop open a door ... or tried to open a bottle of wine without a corkscrew.

Improvisation is critical to 'getting things done'. Sometimes we have exactly the right tool to hand, but often the particular circumstances are not totally foreseen and we need to work with what we have to hand.

We see the same process of appropriation with digital technologies. Email is intended as a way to communicate with others, but some people email themselves web links whilst browsing instead of using the browser's bookmark facility 'communicating' with themselves and others use email attachments as an easy way to share files with a colleague on the next desk.

These improvisations and adaptations around technology are not a sign of failure, things the designer forgot, but show that the technology has been domesticated, that the users understand and are comfortable enough with the technology to use it in their own ways. At this point we know the technology has become the users' own not simply what the designer gave to them. This is *appropriation*.

Appropriation may occur where there is no existing tool for the task, for example, the user who emails themselves the web link because the bookmarks and email folders are separate and they want to organise them together. It may also occur where there is an alternative method, but the appropriation is easier either at the moment or because of learning time, for example, using the email for sharing files instead of configuring shared network folders.

Advantages of appropriation

Appropriation is important for several reasons:

situatedness – In Chapter [\[**5**\]](#) we introduced the notion that design is about *intervention*, not just an artefact or even the artefact and its immediate ways of interaction with it, but the way in which it changes the environment in which it is set. While each word processor may be the same, each office, home or laptop on a train is a different environment. We cannot expect to be able to understand each environment fully and to fill every possible task or need.

dynamics – Environments and needs change. Suppose we designed specifically for a particular work group in a particular office, and covered all eventualities for them. A month later, a year later new people would have joined the group, the external business environment may have changed their focus, they may have bought additional software or furniture that changes their digital and physical workspace. Design for use must be design for change.

ownership – With appropriation comes a sense of ownership. This may be simply a feeling of control, users feeling they are doing things their own way. However, it may also be explicit; often people proudly show you the ways in which they have made use of software and technology to achieve their purposes. As we discussed in Chapter **[**affect**]** these positive feelings can be as important as the things that are achieved.

Sometimes appropriation can be a form of subversion, deliberately using something in a way it was not intended, not just because of something the designer didn't think about, but in order to thwart its intentions. For example, (in the days before mobile phones were ubiquitous, people would often avoid paying the charge on a public payphone by saying something like: "when I'm ready to be picked up I'll ring twice on the phone and then hang up". Whether this is an advantage or disadvantage of appropriation depends on who you are!

This form of subversion is often seen in work contexts, for example, a salesman might deliberately create a 'phantom order' and later withdraw it in order to ensure there is stock available for a loyal customer. In this kind of setting the subversion is against the formal rules in the system, but may be working towards the same ultimate goal of making the best sales for the company.

Design guidelines for appropriation

The idea of designing for appropriation almost seems like an oxymoron. It is like saying "plan for the unexpected". However, whilst you cannot design for the unexpected, you can design so that people are more likely to be able to use what you produce for the unexpected – *they* do the final 'design' when the need arises.

In fact that design for appropriation is possible is made most clear by realising that some sorts of design make appropriation difficult or impossible. Consider an espresso machine. It is so special purpose it is hard to imagine any alternative use. (Although human ingenuity is such that we expect a rush of emails telling us about alternative uses for espresso machines!). This also shows that design for appropriation is not always what is desired, the espresso machine does one thing very well indeed – why do any more with it.

However, explicit design guidance to allow appropriation are less clear and a matter of active research. Here are some guidelines to start, but they should be taken as ways to make you think, not a tick list to verify.

allow interpretation – Don't make everything in your system or product have a fixed meaning, but include elements where users can add their own meanings. For example, in MacOS you can associate colours with files. However, there is no fixed meaning to a red file. It might mean urgent, or a problem, but it is the user who provides the interpretation, the system just colours it. Similarly, in MacOS folders and Windows desktop you can position file icons freely. The location does not mean anything to the system and so again you can use this to group files; perhaps having one corner of a folder window means "finished with". If you are designing a database system, you might consider including a

free text comment field so that users can add details you haven't thought of, just like they might write comments on a paper form.

provide visibility – Make the functioning of the system obvious to the users so that they can know what the effects of actions are likely to be and so make the system do what they would like. This is particularly important when the effects of actions are distant or at different time, for example in a collaborative application. Often system, particularly networks, try to cover up or hide the underlying 'details'. This is fine so long as it is totally and permanently hidden, but often the details leak out, for example, at the limits of wireless coverage. However, users can find their way round these problems if they are made clear, for example, signal level bars on a mobile phone allow you to look for better signal. *Seamful design* deliberately exposes these 'seams' in coverage and connectivity in order to create games and applications.

expose intentions – While appropriation can be very powerful, we have also seen that it can be used to subvert systems. Rather than trying to prevent such subversion the designer can deliberately aim to expose the intention behind the system. This means that (cooperative) users can choose appropriations that may subvert the rules of the system and yet still preserve the intent. For example, if logging into a system is a slow process, then one member of a work group may login once at the beginning of the day and then everyone use the logged in system. If the purpose of the login is security, then this may be fine so long as the machine is never left unattended, however if the purpose is to adapt the system to individual users then this appropriation would be inappropriate. Exposing the intentions behind a system can be a frightening thing – you cannot hide behind "well that's the way it works". However, doing this can make the assumptions explicit and if they are wrong then they need to be re-examined. In the login example, if the purpose is personalisation would it be possible to allow a single secure login but have a facility to quickly swap between users.

support not control – As a designer you want to do things right, to make them as efficient as optimal as possible. However, if you optimise for one task you typically make others more difficult. In Chapter [\[\[**task**\]\]](#) we looked at task analysis. In some situations, such as very repetitive tasks, then designing explicitly for the task may be the correct thing to do, perhaps taking the user step by step through the activities. However, more often the tasks description is incomplete and approximate, in particular ignoring exceptions. Instead of designing a system to *do* the task you can instead design a system so that the task *can be done*. That is you provide the necessary functions so that the user can achieve the task, but not drive the user through the steps. Dourish describes this as "informal assemblage of steps rather than rote procedure driven by the system" [\[\[**Dourish-p160**\]\]](#). Of course you still want the common tasks done efficiently and so you may provide fast paths, or wizards to perform frequent activities using the basic tools and operations ... remembering of course *visibility* making sure the user understand what is being done.

plugability and configuration – Related closely to the idea of support is to create systems where the parts can be plugged together in different ways by the user. Quoting from Dourish again "Users, not designers manage coupling" [\[\[**ibid**\]\]](#). This is most obvious in programmable or scriptable systems and

there is considerable work in making these more accessible to the user. MacOS provides a system called Automator that allows you to chain together small actions on different applications, so that, for example, you can create a workflow that takes a collection of images, sepia tints them and then mails them all to a group of people from your address book. Musicians will also be familiar with wiring diagram like interfaces such as **** which allow MIDI outputs and inputs to be connected together.

[[** screen shots of music interface and/or apple automator **]]

DESIGN FOCUS

Applying design for appropriation

To see how *pluggability* might be worked out in practical design consider the Phillips **not sure of the name** television that projects colour into the room depending on the picture content. The idea is to grow the TV image almost through the room, but is for that and that alone. Internally there is quite likely a fairly loose coupling between the processing that works out the colour on the screen edges and the input to the lights around the edge of the TV. Imagine if this were made 'soft' so that the user could choose to send other signals into the lighting elements, perhaps the stereo sound signal so that the volume increase the brightness and the pitch controls the hue. If the TV is also connected to the household PC, then enthusiasts might create downloadable Internet applications (*encourage sharing*): for example mapping weather forecasts to light patterns, so that the TV could function as an *ambient display* when not in use.

[[**image of Phillips TV **]]

encourage sharing – People are proud of their appropriations of technology, so let them tell others about it. If one user learns a good trick for using your application or device, then this may be useful to others as well. As we saw in Chapter [[**help**]] documentation can be enhanced by end-user contributed material and there are many web sites offering tips and advice on different software. However, you could consider designing this as an integral part of your system, perhaps a 'tips' button that allows users to annotate functions with their favourite tricks. This could function across institutions making use of the *communities of practice* (see Chapter [[**org**]]) to which the user belongs. This sharing is even more important in the case of programmable systems. Even if the scripting or configuration is designed to be 'easy' it will still be only a small subset of users who actively script. However, if you make it easy for those more confident or more technically adroit users to share with others then your product grows all on its own. The MacOS Dashboard is a good example of this as is the Firefox architecture. Both allow the creation of plugins using a combination of small XML and HTML files and Javascript, but importantly in addition have web sites dedicated to sharing these.

[[NOTE - make sure communities of practice is there!]]

learn from appropriation – After a while one old, but broad bladed screwdriver becomes 'the' paint tin opener. What was once a temporary use of a tool has become specialised. This crystallising of appropriation leads to a new tool and the entrepreneur might spot this, notice the particular kinds of screwdriver that made good paint tin openers and then design a sell a special purpose tool just

for the job. By observing the ways in which technology has been appropriated we may then redesign the technology to better support the newly discovered uses. This is a form of co-design where the users are considered an integral part of the design process. This closing of the *Technology Appropriation Cycle* has been called *design from appropriation* [\[\[**design-in-use-ref**\]\]](#). Of course any redesign should also take into account potential further appropriation. This learning from appropriation is particularly easy in some web applications (e.g. blogging or photosharing sites) where the results of users appropriation of the application are easily visible to the designers.

A common feature to these principles is openness, making things that allow themselves to be used in unexpected ways. It is also to some extent about humility, knowing that you do not understand completely what will happen in real use, no matter how good your user centred design process has been.

DESIGN FOCUS

Snip!t – Design from appropriation

Snip!t is a web bookmarking / notetaking tool designed by one of the authors. The aim was to allow collections of snips of material to be collected and organised and then viewed from any web browser. One of the earliest uses however frequently put pieces of text into a local HTML file, then snipped this. She then moved to another machine and viewed the snip in order to copy the text. Effectively she was using the bookmarking tool as a between machines copy-paste. Once this behaviour had been highlighted the system was extended to allow the user to create a snip using a web form not just from an existing page. This extension supported the newly discovered behaviour, but is itself quite an open mechanism with potential for further appropriation.

[\[\[**image **\]\]](#)