# 11
# Usability for Hackers

The heart of software craftsmanship is writing code that is usable to programmers, including the author re-reading his own code in six months.

In this chapter we will look at usability concerns that make a user interface make sense to non-programmers. Certain kinds of math loom large in algorithms, and cultural anthropology looms large in good usability. We will look at anthropology as it relates to usability, and move from theory to practice.

In this chapter we will discuss:

- Anthropology as the foundation of usability
- Anthropologists' strengths that Django hackers are likely to share
- An example of an anthropology-based technique and the surprise it brought an investigator
- Why focus groups are deprecated for usability research
- Anthropology as a way of breaking up a seemingly simple and straightforward thing and understanding it through multiple cultural explanations
- How to apply this foundation to observing users as they deal with an interface
- How good usability observation is like solving a hard bug
- Examples of other background elements we can turn to our advantage in usability observation
- Liabilities that this book's readership may need to compensate for
- What the main tools for usability testing are and when they should be used
- Usability as the soul of what is Pythonic
- Further reading and resources

This chapter takes a little step further back than some of the others, and there is a lot there, but it is interesting material. Let's dig in!

# Usability begins with anthropology… and Django hackers have a good start on anthropology

If you're reading this text, there's a good chance that you are already halfway to being an anthropologist.

Note: for the purposes of this chapter, 'anthropology' is used to refer to cultural anthropology. Other anthropological disciplines exist (for example, biological or physical anthropology, linguistic anthropology, and archaeology), but it is cultural anthropology and its techniques which are most directly relevant here.

How could an author know that you are probably at least half an anthropologist? Let's turn the question around. Why are you reading this book at all? **Visual Basic .NET** has enormous marketing muscle behind it, possibly eclipsing the marketing budgets for all open source technologies put together. Guido van Rossum holds a dim view of marketing, as does much of the Python community. **Monster.com** lists three thousand Visual Basic positions, almost five thousand .NET positions, but only one thousand Python positions. Why are you reading a Django book when you could be reading a title like *Completely Master Visual Basic in Thirty Seconds or Less*?

You are probably a hacker. It does not matter if you were mortified when you found out the preferred JavaScript technique to create an object with fields that aren't globally accessible variables, or if you wince when you hear of a devious way to get technology to do things that shouldn't be possible, or if you have no desire to be considered a 133t hax0r. You're probably a hacker. The classic **How to Become a Hacker** (`http://www.catb.org/~esr/faqs/hacker-howto.html`) for the most part outlines things that have a very obvious relationship to being a hacker: attitudes towards technical problem solving, or learning an open source Unix, learning to program and contribute to the web, and so on and so forth. Towards the end there is a particularly interesting section, because on the surface it looks completely beside the point. The section is titled **Points for Style** and mentions learning to write well, reading in science fiction, training in martial arts, meditation, music (preferably obscure), and wordplay. Other things could be added: avoiding mainstream TV or having arcane hobbies and interests, for instance, so that in a social context hackers may ask each other questions about obscure hobbies as a rough social equivalent to, "What's your favorite TV show?"

Not that any of these is necessary to be a hacker, but together these common trends point to a personality profile that can learn the anthropological style of observation relevant to usability work much more easily than the general public, or even Joe Professional Programmer who regards learning new technologies as a necessary evil rather than a joy, works in Visual Basic .NET after being swayed by advertising, goes home and watches TV after work, has probably never heard of ThinkGeek, and would probably rather do gift shopping at Walmart even if that programmer does know of ThinkGeek.

All of this is to say that the culture surrounding you is not, to you, like water to a fish. It is a basic fact of life that you don't automatically share the perspective of others. Cross-cultural experience or ethnic minority status may accentuate this, but this is true even if you're not (regarded as) a minority. And this kind of experience provides a very good foundation for anthropological ways of understanding exactly how you are not a user and users don't think like you.

# Anthropological usability techniques

There are a number of anthropology-based techniques that apply in usability. They include card sorting, discussed below, and also activities such as observing users try to use your site, eyetracking "heatmap" studies, cognitive walkthroughs, and so on. `http://www.usabilityfirst.com/usability-methods/` shows some standard techniques, although it lists focus groups without the standard warning. Focus groups, as will be discussed, are cargo cult anthropology. They are considered harmful when used for usability instead of marketing research, and are for that matter considered harmful by some of the more savvy market researchers who are realizing that to understand your audience you need the best anthropology has to offer.

# An introductory example: card sorting

One basic challenge for organizing a site's information architecture is the taxonomy, or way of breaking things down. If one is asked for an example of a good taxonomy, one example par excellence is the biological taxonomy that organizes all the way from kingdoms down to species or subspecies and varieties. And indeed that is one kind of taxonomy, but it is not the only possibility. If one is asked to break down a list of a fork, spoon, plate, bowl, soup, and macaroni and cheese, one obvious way is to put the fork and spoon together as cutlery, the plate and bowl together as dishware, and the soup and macaroni and cheese together as food. But this is not the only basic way, and it can make sense to put the fork, plate, and macaroni and cheese together as representing one complete option, and the spoon, bowl, and soup together as representing another basic option.

Stores and websites that have adopted the latter approach, such as a gardening store or website that organizes its products according to the type of garden a customer is trying to make and what the customer is trying to do, see a significant increase in sales. Even biology could use other complementary technologies: a taxonomy that classified organisms according to both ecosystems and their roles within their ecosystems and ecological subsystems could say something very valuable that the eighteenth century classification wouldn't.

In terms of websites, an information architecture that corresponds to the organization's org chart is never a helpful choice. Even when we are talking about an intranet intended only for organizational insiders, one section or subsite for each department is not the right choice. A better option would be to support workflow and design around the tasks that employees will be doing with the intranet.

What is the best information architecture? That is not a question to answer by looking something up in a book or even thinking it out. It is something that we should work out based on what we observe doing research, even if we also read and need to do a bit of thinking. And this is the best practice across the board for usability.

One valuable exercise to help guide information architecture design is called card sorting. In this exercise, we get a stack of index cards, perhaps 3 x 5, and write the individual names of different pieces of functionality the website should offer, trying to name things neutrally so that the names do not have common terms suggesting how certain parts belong together. Then we shuffle and lay out the cards, and individually ask subjects (people who will participate in an experiment and who are not insiders, whether employees of your organization or of an external website, or information technology professionals) to organize them so that cards that belong together are put in the same stack.

Then we note which cards have been placed together, thank the subject, and move on to the next person.

On looking through the notes, we may see a few things. First, not all people think the same. We will likely see some breakdowns that are very similar, but there will likely be two or more breakdowns as fundamentally divergent as our breakdowns of the fork, spoon, plate, bowl, soup, and macaroni and cheese. Second, there will probably be a breakdown that simply catches us off guard. And this is good; it means the exercise is working.

After doing this, we can go about looking for a preferably standard information architecture that will gracefully serve the major ways we observed of breaking things down.

# Focus groups: cargo cult research for usability

With an eye to how to best approach observation, we would like to take a moment to talk about Coca-Cola's blunder with "New Coke" and explain why focus groups—bringing in a group of people and asking them what they want—are deprecated as a recipe to make products that look good on paper but don't wear well in normal use. For those of you who don't remember the uproar some years back, the Coca-Cola company announced that it was switching to a new and improved formula, and there was massive public outlash from people who wanted the old Coke back. (Now the company sells both the old formula as Coke Classic and the new formula as Coke II, and Coke Classic is vastly more popular.)

Why would the Coca-Cola company announce it was terminating its cash cow? The answer is that it did naïve marketing research, ran taste tests, and asked members of the public which they would choose: the formula today sold as Coke Classic, or the formula today sold as Coke II. The rather clear answer from the taste tests was that people said they would rather have the new formula, and it was a clear enough answer that it looked like a sensible course of action to simply drop the second-best formula. It was not until everybody could see that the Coca-Cola company had given itself a PR black eye that the company woke up to a baseline observation in anthropology: *the horse's mouth is a vastly overrated source of information*. Most anthropological observations, including the kinds relevant to usability, are about paying close attention to what people do, and not being too distracted by their good faith efforts to explain things that are very hard to get right.

# Anthropological observation: the bedrock of usability

The first step in understanding your users is to wake up to how something that goes without saying to the entire development team can be unknown to your users. The first step to that is waking up to how many ways there can be to see one clear, simple situation.

# More than one way to see the same situation

The kind of observation needed is probably closest to the anthropological technique of participant observation, except that instead of participating in using software or a website, we are observing others as they use software. Half the goal is to understand how the same thing can be observed differently. To quote from James Spradley's *Participant Observation*, which is an excellent resource:

> *One afternoon in 1973 I came across the following news item in the Minneapolis Tribune:*
>
> > *Nov. 23, 1973. Hartford, Connecticut. Three policemen giving a heart massage and oxygen to a heart attack victim Friday were attacked by a crowd of 75 to 100 people who apparently did not realize what the policemen were doing. Other policemen fended off the crowd of mostly Spanish-speaking residents until an ambulance arrived. Police said they tried to explain to the crowd what they were doing, but the crowd apparently thought they were beating the woman.*
> >
> > *Despite the policemen's efforts the victim, Evangelica Echevacria, 59, died.*
>
> *Here we see people using their culture. Members of two different groups observed the same event but their interpretations were drastically different. The crowd used their cultural knowledge (a) to interpret the behavior of the policemen as cruel and (b) to act on the woman's behalf to put a stop to what they perceived as brutality. They had acquired the cultural principles for acting and interpreting things this way through a particular shared experience.*
>
> *The policemen, on the other hand, used their cultural knowledge (a) to interpret the woman's condition as heart failure and their own behavior as life-saving effort and (b) to give her cardiac massage and oxygen. They used artifacts like an oxygen mask and ambulance. Furthermore, they interpreted the actions of the crowd in an entirely different manner from how the crowd saw their own behavior. The two groups of people each had elaborate cultural rules for interpreting their experience and for acting in emergency situations, and the conflict arose, at least in part, because these cultural rules were so different.*

Before making my main point, we would simply like to comment that the Spanish-speaking crowd's response makes a *lot* more sense than it would first seem. It makes a lot of sense even on the assumption that the crowd did in fact understand the police officer's explanation that they "apparently did not understand." What the article explicitly states is that the police officers were using an oxygen mask, and that is a device that needs to be pressed against a person's face and necessarily cover the same parts of a person's face one would cover to try to cause suffocation. If you are not expecting something like that, it looks awfully strange. (At best!)

Furthermore, although we might not know whether this actually happened, it is standard operating procedure to many emergency medical technicians and paramedics who perform CPR to cut off the person's top completely, palpate to the best place to place one's hands, and mark the spot with a ball-point pen. This may or may not have happened here, but if it did, it is appropriate enough for neighbors to view it as an extreme indignity. Lastly, although today's best practices in CPR are more forceful than was recommended in the past, "heart massage" is a technical term that does not refer to anything like softly kneading a friend's shoulder. People who do CPR regularly say they crack ribs all the time: cracking ribs may not be desirable on its own, but if a responder is doing good CPR with enough force to be effective, breaking a patient's ribs is considered entirely normal and *not* a red flag that CPR is being done inappropriately or with excessive force.

Furthermore, the woman's age of 59 raises the question of osteoporosis. Racism is almost certainly a factor in the community's memories; the community had quite probable stories circulating of bad treatment by police officers and possible police brutality. We know that the police *tried* to explain what they were doing, but if many of us saw police apparently trying to suffocate a member of our community, possibly saw an offensive indignity in that a senior's shirt and underwear had been cut away, and saw an officer keep on forcefully shoving down on her chest and probably heard ribs crackling with every shove, it would take quite some convincing, and almost a reprehensible gullibility, for the bystanders to believe the other officers who tried to explain, "No, really, we're trying to help her!"

For reasons that follow, we should be very wary of saying that she probably would have survived if *only* the crowd hadn't intervened. We shall view this example from a historical point of view, that is, the information presented here was the medical opinion of the times.

We may pause to note that neither group, nor apparently the authors of the newspaper article or anthropology text, appears to grasp how the situation would be viewed by a doctor. "Heart massage" is now more commonly known as "Cardio-pulmonary *resuscitation*" or CPR, *resuscitation* being an otherwise obscure synonym for *resurrection* or returning from the dead. In French religious language, for instance, *resuscitation* is the term one uses for Christ returning to life after death on a cross. There is, to the purist, some fundamental confusion in the marketing-style slogan, "CPR saves lives." Clinically and legally, it was thought that death occurs when a person's heart stops beating. If a person is still alive, and if there is any chance of saving the person's life, then CPR is both premature and inappropriate.

The **Uniform Determination Of Death Act**, United States (1981): An individual who has sustained either (1) irreversible cessation of circulatory and respiratory functions, or (2) irreversible cessation of all functions of the entire brain, including the brain stem, is pronounced death.

Once a person has entered a state of "cardiac arrest," which meant *death* (historically), then there might be a possibility of getting that person back by cardio-pulmonary resuscitation, even if that is a long shot. CPR at its very best is a third as effective as a heart beating normally, and even under ideal conditions can only slow deterioration to give the emergency room perhaps a 5% to 10% chance of getting the person back. And that is assuming that ideal conditions are possible: in reality ideal conditions do not happen. Though most people giving CPR do not have to deal with a crowd interpreting their efforts as assault, hoping to deliver perfect CPR is like hoping to become a good enough coder that one need not contend with debugging. Eric Raymond implicitly showed great maturity as a programmer by saying he was dumbfounded when his first attempt at Python meta-programming worked without debugging. The person who does CPR in a public setting will contend not only with the difficulties of CPR itself, but an "uh-oh squad", bystanders who second-guess one's efforts and create a social dynamic like that of giving a speech to an audience of hecklers.

Now there is no question of blows or physical restraint when it comes to the idea of CPR or cardiac massage as a way to save lives that is apparently shared by the newspaper article author, the anthropology author, and possibly the police. And the medical view is that CPR is "only indicated in the case of cardiac arrest," meaning that it is premature and inappropriate unless a person has already died, but can preserve a remote chance of getting a patient back after the patient has crossed the threshold of clinical death. Emergency room doctors who view CPR as slowing deterioration and holding onto a slender chance of getting someone back will be quite grateful for CPR performed by police officers and other members of the general public who view CPR as a skill which saves lives. But the understanding is still fundamentally different, and differences like this come up in how computer interfaces are understood: differences you will want and need to appreciate.

# Applying this foundation to usability

The core of usability testing is designing some sample tasks, asking users to do them, and observing, as a fly on the wall, *without helping*. If you can record sessions, great; if not, a notepad, notebook, or netbook works well. (The advantage of recording sessions is that almost invariably people will say, "There is no way the user could have that much trouble with our design," and a five-minute video of a user looking everywhere on the page but where users are intended to look, is worth a thousand arguments.) Usually studying five users is sufficient.

There is a saying in customer service of, "The customer is always right." One may read the cautionary tale of a salesperson who kept on winning arguments with customers and somehow never closed a sale. And the principle is very simple. A customer who is wrong is to be treated as a valued customer as well as a customer who is right, and whether our customer is right or wrong, we treat each customer as a valued customer. Unless we are talking about an abusive customer, in which case it is appropriate to draw a line in the sand, we don't send a message of "I'm right, you're wrong."

That is *not* what we are talking about when we say, "The user is always right." Anyone who teaches programmers or remembers what it was like to begin programming remembers hearing, "There's no way the computer can be right! The computer has to be running my code wrong, or the compiler isn't working right, or SQL's lying to me again!" And it is a slow and at times painful lesson that the computer is in fact (almost) always right, that no matter how right your code seems, or how certain you are, if your code is not working, it is because you did something you did not intend, and your code will begin working when you find out how your code does not obviously say what you think it does, and adjust that part of your code. Bugs in libraries and (more rarely) compilers and interpreters do exist, but one important threshold has been crossed when a programmer stops blaming the tool for confusing bugs and begins to take responsibility personally.

And in the same sense that the computer is always right, and not the sense that the customer is always right, the user is always right about how users behave. If the user interacts with the user interface and does something counterproductive, this means the same sort of thing as code doing something counterproductive if it has been compiled. The user, who is always right, has identified an area where the interface needs improvement. The user should be regarded as "always right" just as the computer should be regarded as "always right," and when the user is wrong, that is good information about where the user interface has problems.

I could say that the only thing we really need to do at all is observe the user. But observing the user includes a major challenge: it includes the major task of grasping things that violate our assumptions. The task is something like first encountering how JavaScript's support for object-oriented programming includes objects and inheritance, but without classes, first coming to a scripting language and asking, "When does integer overflow occur?" and being told, "Your question does have an answer, but it matters less than you might think," or the experience of a novice programmer who posted to a forum, "How do I turn off all the annoying compiler warnings I'm getting?" and was extremely frustrated to have more than one guru say, "You want to beg your compiler to give you as many warnings as you can get, and treat all warnings as errors."

It was a deft move for Google to give Chrome a single search and URL bar, but the main reason may not be the one you think. Searching was heavily enough used that Firefox made life easier for many users by adding a second bar to the right of the URL bar so that we could search without first pulling up the Google homepage. For heavy users, simplifying the URL bar and the search bar into one full-width piece is the next refinement. But this is not the main reason why it was deft for Google to give Chrome a unified search/URL bar, or at very least not the only reason.

My own experience helping others out with their computers has revealed that something obvious to us has been absolutely non-existent in their minds. Perhaps you have had the experience, too, of telling someone to enter something in a page's text field, and they start typing it in the URL bar, or vice versa typing a URL into a page's search field. What this unearths is that something that is patently obvious to web designers is not obvious to many web users. "Here is an important, impenetrable dividing line, and all the chrome above that line belongs to the browser, and everything below that line (above the bottom chrome, and excluding any scroll bars) belongs to the website." This division of labor is obvious enough to most web designers that only experience could teach them that there are some people who don't understand it. But the real world has many users who do not have any such concept, and behaviors like typing search terms in the URL bar (years before Chrome was available) are clues to "This is something that is out there."

And if you think, "OK, but users are more sophisticated now," you might go through your website's search logs and see how many website addresses you can see. It will not be nearly as many as ordinary search terms, but have you ever wondered where the addresses to MySpace and porn sites in your search logs come from?

Culture shock is a fundamental reality of when things go contrary to your expectations. Most of us experience small amounts of culture shock in our day-to-day living and much greater amounts if we travel to another country or do something else. The three examples given earlier, of classless objects in JavaScript, integer overflow in scripting languages as not terribly important, and asking for a more draconian handling of warnings are examples of culture shock in relation to technologies. As a rule of thumb, if you are not experiencing culture shock from your user observations, you are not deriving full benefit from them, and you do not understand your users well enough to make the fullest improvements to the design. To put it more forcefully, if you aren't experiencing culture shock from your user observations, that's because you're taking a shower with your raincoat on.

# It's just like (hard) debugging

We would like to make one closing parallel to debugging. There are several types of debugging we are not talking about; for instance, a missing close parenthesis causes an immediate error that makes it fairly quick work to find out what is wrong and what line of code it is. A traceback can also provide an excellent starting point for quick and effective debugging. Although debugging a failed unit test may not be quite so easy, a unit test is not just a tool to say that something is wrong, somewhere; it is a tool that should point a finger, and usually narrow the search field significantly. And many other bugs that are neither syntax errors nor resolved with the help of unit tests are still easy enough to fix that we need not be terribly aware of them. When we think of debugging, we may only think of the few hard bugs rather than the majority of bugs which better programmers resolve without really thinking about it, like we turn on light switches on entering a darkened room, or unzip a coat outdoors when the day warms up, without giving the matter too much conscious thought or vividly remembering that we do this. (This is, incidentally, somewhat of an ethnographic observation of good programmers.)

What we are talking about, as hard bugs, are bugs where you go through every investigative tool you can think of, and still cannot pin down what is going on. (This may include a relatively small proportion of bugs that also generate tracebacks or unit test failures.) Observing the bug seems like observing, not a miniature ship in a bottle, but a ship in a seamless glass sphere. There's no way you can tell that the ship could have gotten in there, but it is quite clear that the ship in fact is in a glass container that has no openings that you can imagine the ship getting in through.

Isaac Asimov said, "The most exciting sound in science is not, '*Eureka!*' [I've found it!], but 'That's funny,'" and the history of science bears him out. Today, X-rays are widely known among scientifically literate people to be a very high-energy, short-wavelength radiation belonging to the same spectrum as visible light, but it was not always so; the name "X-rays" is itself a holdover from when they were a fascinating and mysterious phenomenon, with the "X" in "X-rays" referring to something unknown. It was known that they were radiation of some sort, but they passed through some opaque material and in general did not fit into anything people had a conceptual place for.

In the middle of efforts to understand this mystery, there was one physicist who stumbled upon a golden clue that X-rays might be something like light He left unexposed photographic plates near a source of X-rays, and upon using and developing them, observed that they had all been partially exposed. His response, however, was to contact the photographic supply company and demand that they replace the photographic plates as defective. As Winston Churchill observed, "Man will occasionally stumble over the truth, but most of the time he will pick himself up and continue on."

In debugging, hard bugs, the kind that remain unresolved after we have investigated all the usual suspects, are rarely solved because we go looking for the right error and find exactly what we expected to find. With the analogy of the ship in the sphere, it is more like deciding there has to be some kind of concealed seam from gluing or otherwise sealing an aperture big enough to allow the ship to enter, at least in pieces. After looking the glasswork over, using magnifying glasses and lights, and still finding no trace of a seam, you stop ignoring something you had noticed along the way: the ship itself appeared surprisingly glossy. When you stop to look at the ship for a second, you realize that it is not made of the wood and cloth you expected (and that it appears to be at first glance), but as far as you can tell is shaped out of colored glass. And, after doing a little more research, you learn of a glassblower who makes colored glass ships and forms seamless glass spheres around them. In this case, you were not wrong in saying there was no seam. There is still no way that such a thing could have been crafted at room temperature, and there is in fact no ultra-subtle seam that you failed to notice in your efforts to find the seam to an aperture through which the ship could have been inserted at room temperature, even in pieces. But that's not the point. The ship in a globe was made at glassblowers' temperatures, and there it is possible to create a seamless sphere around a colored glass ship.

Hard bugs are debugged successfully when you learn to stop upon stumbling over the truth. And the same is true in the anthropological side of usability techniques: some things you can know to look for, and find, but the much more important competency is to recognize when you have stumbled over the truth, and stop and pay attention to something you don't know to look for.

Almost all of the difference between doing user observation badly and doing it well hinges on learning to recognize when you have stumbled over the truth.

# Lessons from other areas

Opportunities to learn, and sharpen, core anthropological competencies for usability are all around us. We need, in many cases, simply to open our eyes, *and transfer what we have learned from one area to another.*

# Live cross-cultural encounters

Learning and observing in cross-cultural encounters is an excellent way to learn how to pick up cues the way a user interface developer needs to. There are two basic cross-cultural encounters we recommend as particularly valuable. The first of these, as it takes shape in the U.S., is to spend time volunteering with an English as a Second Language program and tutor on computer basics. Or find out if you can tutor in classes at your local library. (If possible, work in an adult computer class that has seniors and not too many young people.) This may or may not be the most

pleasant experience, but it is one of the most valuable. I remember one experience where I was working with a Sudanese refugee, quite possibly an escapee of genocide against Christians, who had just had his life uprooted under presumably traumatic circumstances and was learning to deal with living in the U.S. all at once, which would quite probably be traumatic in itself. I remember, in particular, one moment when we had very slowly typed a word or two in a word processor, and ticked the button to close a document, and were staring at a dialog box asking if we wanted to save the document before closing. And I remember a slow dawning realization that not only did he not know the quite substantial cultural concepts involved in recognizing that this was how culturally one asks a question, expecting an answer in the form of a click on one of two areas of the screen to answer **Yes**, **No**, or "*Mu*" (**Cancel**). But the question itself, "Do you want to save this document before closing?" was a question that did not exist at all in his culture, and even if I spoke his native language I would probably not be able to explain the question in terms that would make any sense to him. That was probably my most difficult teaching experience, and the one where I have the most doubts about whether I succeeded in teaching anything at all. But it was a profoundly valuable experience to me, and helped me see how things could "go without saying" to me but be baffling to others.

The second of these two cross-cultural encounters is whatever you already have. Few if any of us have no cross-cultural encounters; whether one is ethnically or (a)religiously a majority or a minority, an immigrant or a native citizen of one's country, or considering face-to-face encounters or Internet connections, most of us have at least some experience in cross-cultural encounters. The differences are there; if you have learned something from cross-cultural encounter, the experience can help you more readily recognize the cues you need to recognize.

# History

While we are wary of reducing history to merely an apparatus to understand the cultures of previous times, most historians arrive at a fairly deep understanding of a culture that is not their own, and may arrive at a sensitivity to the ways, all too easy to ignore, in which historical texts veto modern assumptions. There was an experiment in which a question concerning Abraham Lincoln and a number of historical primary sources was given to a number of elementary school teachers, plus one historian of Lincoln, and a historian whose specialties were unrelated. During the time of the experiment, the elementary school teachers started with a wrong conceptual framework that imposed today's basic categories on the texts, and did not progress to anything better.

The historian of Lincoln started with a highly accurate conceptual framework and very quickly arrived at the answer. But what is particularly interesting is the other historian, who was trained as a historian but had little directly relevant knowledge of Lincoln. He started with the same conceptual framework as the non-historians, but by the end he had corrected his framework to the point of reaching where the Lincoln historian had started.

This latter historian is perhaps the most interesting, not because he was not initially right, but because he was self-correcting. Even though his starting framework was no better than the school teachers, he was sufficiently able to adjust his perspective from cues based on the text so that he reached the framework the Lincoln historian started with. And, one would imagine, the Lincoln historian would have had a similar self-correcting sensitivity to the texts had he been asked the same kind of question about a historical setting he did not initially understand.

Getting history right is relevant to us in two ways. First, you or I understand one, or perhaps many, other cultures more or less well. Second, when you or I trip over a clue that is wrong, we can stop and learn from it, instead of hoping it will go away. Both of these strengths are a powerful foundation to usability.

# Old books and literature

Books can be a very good place to sharpen anthropological competencies through meeting other cultures. However, we might clear the ground of some distractions if it is tempting to say, "But I meet other cultures in all my favorite books! I'm an avid reader of science fiction and fantasy."

All science fiction is not created equal in terms of cultural encounter. There is a marked difference between reading Heinlein's *Stranger in a Strange Land* and watching *Star Trek*. Heinlein understood both culture and culture shock, and though his book only treats one alien culture, it is written to create culture shock in the reader, and challenge us in assumptions we didn't know we had. "*Whaaa – ?* They can't *do* that!" is a normal and intended reaction to several parts of the book. In *Star Trek*, there are many races, but culture shock in the viewer is almost non-existent even when the plot is intended to surprise. To put it more pointedly, the average American's culture shock from watching years of *Star Trek* is probably much less than the average American student's culture shock from a few months' experience in a foreign exchange program, perhaps less than the culture shock in the first month of that program. By comparison with a live encounter with another human culture, the alien races in *Star Trek* have less their own alien cultures than a shared personality profile we can already relate to even when we don't like it.

Likewise, not all fantasy is created equal. J.R.R. Tolkein and C.S. Lewis were both Oxford-educated medievalists who knew medieval literature intimately. The genre of fantasy that appeared in their wake, if you have seriously read medieval literature, seems by comparison like the opening rant in the movie *Dungeons & Dragons*, where a supposedly medieval character gives an impassioned "Miss America" speech about how horrible it is that the realm's government is unlike a U.S.-style democracy. Today's genre fantasy reads like the story of Westerners from our time who happen to be wearing armor. By contrast, in *The Chronicles of Narnia* some of the characters are indeed from the twentieth century, but in terms of how the story is put together there is something a bit medieval, and not individualist, about their characterization.

If our cultures' science fiction and fantasy are not the best place to be challenged by another encounter, and to develop that kind of sensitivity, where can we go? One obvious response is to look to be challenged by books like the *Dao De Jing* and the *Bhagavad-Gita*. Those are both excellent places to look to be challenged, but if we assume that we can be challenged by the *Bhagavad-Gita* but not Plato, we are selling both of them short. Plato's image of climbing out of the cave with its shadows and looking at the sun is something that a Hindu commentator on the *Bhagavad-Gita* can quite easily relate to, and in a certain sense Plato has more in common with that kind of Hinduism than with his disciple Aristotle.

How does one read a text to see what one can pick up culturally? Consider the following text:

> *QUANTUM THEORY, THE. As recently as the opening years of the present century the vast majority of physicists still regarded Newton's dynamical laws as something established for all time. And they were not without solid grounds for this faith. Many phenomena were indeed known, chiefly those which may be classed under the heading radiation, for example, black body radiation and line spectra, which refused to accommodate themselves to any sort of theory founded on Newtonian principle. But it was generally believed that such phenomena would, sooner or later, be completely accounted for without any departure from the classical principles of physics. Even the theory of relativity developed by Lorentz, Einstein, Minkowski and their successors was regarded only as a widening or generalization of the Newtonian basis of physics. It was the culmination of classical physical theory. These phenomena we now believe, cannot be accounted for on the basis of classical physical theory, whether Newtonian or Einsteinian. The first act of sacrilege was committed by Max Planck, until recently professor of theoretical physics at the University of Berlin, about the end of the year 1900, when he initiated the quantum theory. One of the problems engaging the attention of physicists during the closing years of the last century was that of the radiation from a black body...*

> *The reconciliation of these two aspects of the phenomenon, namely the independence of the energy of the ejected photo-electrons and the intensity, on the one hand, and the wave character of the radiation on the other, constitutes one of the most formidable problems which physical science has ever encountered...*

Now we would like to make a couple of points. We could, for instance, have chosen an interminable fight narrative from a medieval Arthurian legend to say, "We look on Arthurian legends as mysterious tales of wonder. Did you know that a large portion of those legends is actually quite dull to the modern reader?" Some readers may be wondering, "This is a scientific article, not a cultural area where anything goes." But, even if science is not a domain where anything goes, there are cultural issues here, and it may be possible to date the article by cultural markers as well as by values given for physical constants. (Based on details further on in the text, Avogadro's number appears to be given as 6.06 x 1023, not today's 6.022 x 1023, and the unit of electrical charge is reported in the text to have current values consistent with initial measurements, despite the fact that the initial reported experimental value was erroneous and subsequent experimenters fudged until it was found acceptable to report what is now believed to be the correct value.)

In the quoted text, there are two significant markers that date the text as showing significant cultural difference from how things are viewed today.

A physicist or philosopher today would say that Newtonian physics, Einsteinian physics, quantum physics, and for that matter superstring theory are fundamentally irreconcilable on an ontological plane but happen to predict the same behaviors for the kind of experiments one would expect of a high school physics lab: the predicted results for each of these theories are vastly smaller than even a top-notch experimental physicist doing high school experiments could possibly observe. But the reasons behind those differences are irreconcilable, like the difference between saying "You see this OS behavior because it is running natively on your computer," and "You see this OS behavior because it is being emulated under virtualization with several levels of indirection that are extremely slippery to understand." The behavior predicted is interchangeable, but the reasons proposed for the behavior are fundamentally irreconcilable. Furthermore, this is not just true if one compares quantum physics with Einsteinian or Newtonian physics; it is also true if one compares Einsteinian with Newtonian physics: to today's take on things, it is a bit astonishing to say, "on the basis of classical physical theory, whether Newtonian or Einsteinian." The usual way of presenting things in a physics class today is to present Einstein's theory of relativity as the first in a stream of foundational upsets after Newton reigned unchallenged and apparently eternally established for centuries. Today we would expect to need to dig a bit to find more examples of Einstein's theory referred to as a further expansion developing Newton, which should still be considered "classical physical theory."

The second quoted paragraph refers to how light (and, it may be mentioned, practically everything else as seen in quantum theory) behaves as a particle when treated in some ways and as a wave when treated in others. This duality has since hit the rumor mill well enough that a favorite illustration from science in theology programs is how light exists as both a particle and a wave, which reflects the extent to which the duality of light as particle and wave remains unresolved but is no longer regarded as, "one of the most formidable problems that physical science has ever encountered."

Our point is not to deride the article, which is written at a higher level of sophistication and detail than, for instance, Wikipedia. Apart from its certitude in the existence of an "aether," slightly surprising in light of the fact that the Michelson-Morley experiment dates to 1887 and the article refers to 1900 as a past year, its picture of quantum physics portrays the same core science one would expect of a physics text today. But, even in physics, which is not in any sense a field where just anything goes, culture is present, and for that matter in this article the cultural cues alone are most likely sufficient for an historian of twentieth century physics to closely date it.

This kind of cue is what you can practice learning in reading old books, and this kind of cue is what you need to be able to pick up in observing for good user interface development.

The way you observe that a user does not share an understanding that is obvious to you is by the same kind of cue that can clue you in that a text doesn't share an understanding that is obvious to you.

# The last other area: whatever you have

Whatever else you have is probably a resource you can draw on. Do you love birding? Birding is a hobby of observation. Do you do martial arts, for instance? A common theme in martial arts is harmony between opponents, and if you can attune yourself to a sparring partner, you should be able to attune yourself to a user. Comedy or performing arts? You are not a good comedian if you are insensitive to your audience. Have you made a lot of mistakes, and learned from them, or at least *started* to learn? Wonderful news! (Are you an amateur or professional anthropologist? *That* one does not need explaining!) There is some connection between any two areas of life; let other skills support and strengthen your usability work.

# Understanding the user

Do we best understand the user by sitting off by ourselves and speculating how a user would look, think, and act about our interface? Let us take a look at what may be a very familiar lesson to expert programmers, however unreal it may seem to novices.

# A lesson from optimization

Knuth said, for the novice programmer, "Do not optimize," and to experts only, "Optimize later." Always writing for optimization is a recipe for bad, unreadable code, and for that matter slow code, compared to code written for clarity that is later optimized using that clarity. And Knuth also said, "Premature optimization is the root of all evil."

In one production system I was working on, I wrote one search with the realization that the implementation I was using was extremely inefficient, and had to deliberately refrain from optimizing it, to leave for later. When the whole system was put together, it took a couple of seconds longer than was acceptable, and I began mentally gearing up to optimize the inefficient search. Before doing so, I did some testing, and found to my surprise that my inefficient search implementation took very little time to run, and when I began mapping things out, found the root problem. I had called a poorly chosen method, and with that choice made a purely preventable network call, and that network call took a few seconds. When that problem was fixed, the remaining code ran at an acceptably fast rate for even the largest accounts.

This story is my own version of something that keeps on being retold in the programming literature: "Our system was running slowly, and we had reasonable ideas about what was going on here, but our reasonable ideas were wrong. We didn't know what the real problem was until we dug into some observation."

This basic lesson in optimization is a fundamental phenomenon in usability as well. We will have reasonable ideas about what the usability issues are, and our reasonable ideas will be wrong. We won't know what the real issues are until we dig into some observation.

# What's wrong with scratching an itch, or you are not your user

The open source community is largely driven by scratching itches, but scratching a programmer's itch is a terrible way to approach user interface design.

The story is told of a program used in an office where a popup window appeared and said, "Type mismatch." And the secretary obediently typed M-I-S-M-A-T-C-H, a perfectly appropriate user response to an inappropriate error message. (This kind of thing shows up in many more subtle ways, some of which are not so obviously wrong.)

Designing a user interface that makes sense to someone who understands its inner workings, and designing a user interface that makes sense to its intended audience, are not the same thing. A mechanic's understanding of how a car starts is very elaborate and detailed, but a user should be able to get by thinking, "I turn the key and press the gas, and the car starts" without necessarily thinking anything about what is under the hood. If users need to understand what's under the hood to operate the car, the car needs improvement.

# Worst practices from the jargon file

The **Jargon File** defines the extremely pejorative "PEBKAC" as:

> *[Abbrev., "Problem Exists Between Keyboard And Chair"] Used by support people, particularly at call centers and help desks. Not used with the public. Denotes pilot error as the cause of the crash, especially stupid errors that even a luser could figure out. Very derogatory. Usage: 'Did you ever figure out why that guy couldn't print?' 'Yeah, he kept canceling the operation before it could finish. PEBKAC'. See also ID10T. Compare pilot error, UBD.*

And the particular example is unfortunately revealing of an attitude user interface people need to avoid like the plague.

It is common enough in computer programs to have modal dialog boxes; the humble JavaScript `alert("Hello, world!");` is one of innumerable ways to get them. And what they mean from an ordinary non-technical user perspective is, "A box popped up, probably one that you do not want and may not understand. What is even more annoying is that it is blocking your work; you can't continue what you are doing until you get rid of it." And so an entirely appropriate way to deal with these annoyances is to get rid of them as quickly as possible.

The example given in the Jargon File's definition of "PEBKAC" is:

> *'Did you ever figure out why that guy couldn't print?' 'Yeah, he kept canceling the operation before it could finish. PEBKAC.'*

For a long time, at least, attempting to print from a GUI gave something that looked like a modal dialog box, but for this "modal dialog lookalike," there is one important difference in behavior. When you click on the button to make it go away, it destroys your print job. This is not a case of a problem existing between the user's keyboard and chair. It is a case of a problem existing between the user interface designer's keyboard and chair. PEBKAC.

To pick on the jargon file a little more, "Drool-proof paper" is defined as:

> *Documentation that has been obsessively dumbed down, to the point where only a cretin could bear to read it, is said to have succumbed to the "drool-proof paper syndrome" or to have been "written on drool-proof paper." For example, this is an actual quote from Apple Computer's LaserWriter manual: "Do not expose your LaserWriter to open fire or flame."*

Let's ignore the fact that this sounds less like a technical writer trying to be easy to understand, than corporate legal counsel trying to ward off ambulance chasers.

There is a very user-hostile attitude here: the basic idea that if your system is too difficult for your users to understand, the users must be too stupid, and making something user-friendly is a matter of stretching to meet people you should not have to cater to. Stories and terms like this circulate among programmers. We might suggest that terms like these, for your software's audience, are little, if any, better than a racial slur. They reflect an attitude we do not need.

# Python and usability

You do not really understand Python until you understand something about usability as it appears in Python. Usability is the soul of "Pythonic."

# It's not all about the computer!

There is something genuinely different about Python, and to explain it we would like to discuss the advantages of C.

If you want to nano-optimize every ounce of performance you can get, there is little serious competition to C. You can write assembler for different platforms, or write in C++ that is multi-paradigm like Python and have some parts of your program use high-level features like objects, templates, and operator overloading,

while still writing almost unadulterated C for parts that are performance-critical. And the group of programmers that "vote with their keyboards" for using C this way, includes Guido van Rossum, who created Python. The first and canonical Python implementation is written in C, and a Pythonista underscoring the point that Python's switch statement is a very efficient dictionary will explain that Python's dictionary is implemented in tightly optimized C.

But this kind of advantage comes at a price. In the canonical list of ways to *shoot yourself in the foot* in different programming languages, C is "for people who want to load their own rounds before shooting themselves in the foot." In one Python forum, a wannabe 133t hax0r asked how to write a buffer overflow in Python, and a wry Pythonista replied apologetically: "We're sorry, but Python doesn't support that feature." But C does support the "feature" of buffer overflows. Its default string handling never leaves home without it. With manual memory management and manual handling of pointers, C also supports "features" including all kinds of memory leaks and subtle pointer errors that can be extremely difficult to debug. Python closes this Pandora's box, although Python is hardly the only language with the wisdom to do so. Python, PHP, Ruby, Perl, Tcl, and Java all close the Pandora's box that must be wide open if you are to have tightly optimized C.

C has been called a language that combines the power of using assembler with the ease of using assembler, and there may be no compiled language that surpasses C for power over bare metal, or for corresponding possibilities for tight optimization. However, this is not the only way to keep score. Python keeps score by another metric: programmer productivity.

The one overriding concern motivating decisions in Python is not how you can get the tightest control over the computer's productivity. It is how to let the programmer be most productive, and it has been said of this relentless pursuit of programmer productivity that capital sentences are passed with less thorough deliberation than obscure Python features. And if you have used Python, the difference you have experienced is precisely because of this one overriding concern, this relentless pursuit. The people in charge of Python have decided that Python is not about what to do to optimize the computer; it is about what you do to empower the programmer.

If you are interested in usability, you have a good working example of usability to look at. To put Python's strength a little differently, Python is a language where the one overriding concern and relentless pursuit is usability for you, the programmer. If you are working on usability, you are working to give end-users the same kind of thing that Python gives you. You are making a product more Pythonic to use, as opposed to giving the more C-like experience of an interface that lets users load their own rounds before shooting themselves in the foot.

Usability is about how to go from giving C user interfaces, to giving Pythonic user interfaces.

# What to do in the concrete

We might take a cue from expert practice in solving problems updating Gentoo Linux. Gentoo is not an attempt to make the best "for the rest of us" Linux distribution; Gentoo is like a brand of car intended to be a favorite of amateur and professional mechanics. It allows very fine-grained control and tightly tuned systems. However, upgrades do not always work, and if we try to keep a Gentoo installation up to date, we may find that the upgrades break.

There may be a lot to Gentoo, but there is one basic, versatile, powerful technique that works for a wide variety of problems: copy the error message or salient parts, and make a well-crafted search to see where the problem has come up on the web. Most of the time deft searching is the primary skill to address breakage, and when things break, using this one skill well is often all you need even for very different problems.

There are a lot of usability methods, and a web search for "usability methods" will turn up as much information as you care to read. However, one or two basic methods, used well and used persistently, can deliver a lot of power. We suggest:

- Get five "average Joe" users who are not programmers or insiders, and ask them to do tasks with your system. NEVER help them, even by pointing out something very little and obvious. (If we help them, we have tainted our data.) Observe with a notebook; if you can videotape the interactions, that's great. Oftentimes they will have trouble in the "wrong" part of your system, not the part you're trying to focus on; that is still valuable data to heed.
- Observe people in their own "field environment," on their turf, doing their tasks, their ways. Once again they will do things that catch you off guard. Being caught off guard, and writing down what they are doing "wrong," is a good part of why such observations are desirable. This is an even more powerful setting to learn exactly how "The user is always right."

Furthermore, these should not be something tacked on at the end, as user acceptance testing is traditionally done by the waterfall method. They should feed into Agile iterations: *test early, test often*. An early, cheap, unsophisticated set of usability tests early on is more valuable than an expensive and in-depth test when things are set in stone. That the first attempt will be wrong is non-negotiable, and the user interface should be considered wrong until it tests as working—but this is another form of the basic observation in software development that untested code has defects and should be recognized to have defects until it has been tested and defects have been addressed.

# Further reading

There are several resources available to expand your know-how on the topic:

- Academic anthropology texts, even though they usually do not mention usability. James Spradley's *Participant Observation* quoted earlier is one such text. The Wikipedia articles for "participant observation," "ethnography," or "cultural anthropology" provide a good nutshell summary and footnote texts that offer a helpful broadening of horizons.
- With Alertbox (`http://useit.com/alertbox`), Jakob Nielsen provides quick, five- or ten-minute nuggets of usability insight, and while some people complain he is stuck in the past, his work remains a basic primer about things that people still get wrong today.
- Edwin Tufte's works, such as *Envisioning Information*. His works often discuss the process of a makeover to deliver information with more accuracy and less distraction, and are a good reference point. They are not always related to computers, but they deal with usably solving difficult problems in presenting information visually.
- Plain Language (`http://plainlanguage.gov`) is a U.S. government website that deals with makeovers for usability in language. Like Tufte, they start with something that is painful to use and demonstrate how to transform it to something much easier.
- A List Apart, `http://alistapart.com`. Not all of their content is centered on usability; however, they discuss designing things well.
- Web searches for "usability" or related phrases. There's a lot of good stuff there. As mentioned earlier, there are several sites listing usability techniques. One such site was mentioned: `http://www.usabilityfirst.com/usability-methods/`

Only half of these are directly about improving software or web usability, but the combination of direct and oblique references will help you pretty much as far as you have time and energy to dig.

# Summary

We have covered a lot of material here; let us summarize some salient points:

- Most Python/Django types have some personal strengths that are relevant to usability testing, even if we do not realize we have them.
- Cultural anthropology is bedrock to the way of observing that helps with users.
- When you are testing, resist the temptation to help the user and taint the test.
- Waterfall placement of UAT as the last step is broken, but usability works well when it is integrated into Agile iterative processes from the beginning.
- Get it wrong the first time, and keep iterating until it is excellent.
- Test early, test often.
- A lot of unsophisticated tests are worth more than you might think.
- There are some very good resources out there that cover transforming from something terrible to something excellent.
- Focus groups and their whole approach is deprecated in usability work because the horse's mouth is a vastly overrated source of information.
- Observe with a notebook, and record things on camera if you can.
- Especially in usability and user interface, do not ask, "What can the computer do?" Ask instead, "How can the computer support the person using it?"
- Relevant interests that can help us range from history to cross-cultural encounters to martial arts; getting better at usability means stretching out into other areas besides programming.

This brings us to the end of our book. Let us turn to an appendix on debugging. We will look at debugging, and while we will cover tools such as Firebug, our work will not be about tools in the computer, but skills in the developer that wield different tools to cut to the heart of bugs.

Let's begin!