9

Designing for Pleasure

As Albert Einstein said, "You can't solve a problem with the same thinking that created it." I've just devoted many pages to identifying that old thinking and showing how it doesn't work. Now it's time to talk about a new method that *will* work. I've been developing this method, called Goal-Directed design, since 1992, and the designers in my consulting firm use it for all of our projects. It consists of some novel ways of looking at problems, some powerful guiding axioms, and some remarkably effective mental tools. In the next few chapters, I'll present an overview of three of the most powerful of these tools, along with some case studies of how they are applied and the kind of results you can expect.

Personas

The most powerful tools are always simple in concept, but they often must be applied with some sophistication. That is certainly true of interaction design tools. Our most effective tool is profoundly simple: *Develop a precise description of our user and what he wishes to accomplish.* The sophistication comes from how we determine and use that precise description.

The most obvious approach—to find the actual user and ask *him*—doesn't work for a number of reasons, but the main one is that merely being the victim of a particular problem doesn't automatically bestow on one the power to see its solution. The actual user is still a valuable resource, and we devote considerable attention to him or her, but we never let the user directly affect the solution.

The actual method that works sounds trivial, but it is tremendously powerful and effective in every case: We make up pretend users and design for *them.* We call

these pretend users *personas,*[1] and they are the necessary foundation of good interaction design.

Personas are not real people, but they represent them throughout the design process. They are *hypothetical archetypes* of actual users. Although they are imaginary, they are defined with significant rigor and precision. Actually, we don't so much "make up" our personas as *discover* them as a byproduct of the investigation process. We do, however, make up their names and personal details.

Personas are defined by their goals. Goals, of course, are defined by their personas. This may sound tautological, but it is not. Personas reveal themselves through our research and analysis in much the same way that the sequence of tectonic events reveal themselves to geologists through the study of sedimentary layers: The presence of a fossil defines a stratum, and a stratum defines the presence of a fossil. I'll talk a lot about goals in the next chapter, but we discover them in the same way we discover personas. We determine the relevant personas and their goals in a process of successive refinement during our initial investigation of the problem domain.

Typically, we start with a reasonable approximation and quickly converge on a believable population of personas. Although this iterative process is similar to the iterative process used by software engineers during the implementation process, it is significantly different in one major respect. Iterating the design and its premises is quick and easy because we are working in paper and words. Iterating the implementation is slow and difficult because it requires code.

Design for Just One Person

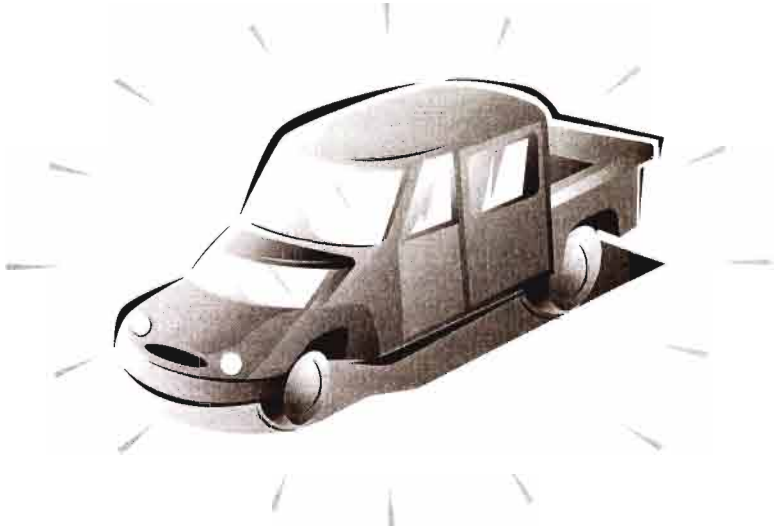
If you want to create a product that satisfies a broad audience of users, logic will tell you to make it as broad in its functionality as possible to accommodate the most people. *Logic is wrong.* You will have far greater success by designing for a single person.

Imagine that you are designing an automobile to please a wide spectrum of people. You could easily identify at least three subgroups: the soccer mom, the carpenter, and the junior executive. Mom wants a safe, stable vehicle with lots of space and big doors for hauling the kids, dogs, groceries, and other stuff. Joe, the carpenter, wants a rugged vehicle with all-wheel drive and abundant room for ladders, lumber, bags of cement, and tools. Seth, the young executive, wants a

[1] *For all of you Latin scholars and typographers out there, you will be happy to know that the battle between "personas" and "personæ" rages hotly and daily at Cooper Interaction Design. Designers on the "personas" side argue that pronunciation is less ambiguous, gratuitous ligatures can be eliminated, and the word appears conventional and unthreatening to our clients. Designers on the "personæ" side argue that the pronunciation is easy once you hear it, the opportunity for a gratuitous ligature is like manna from heaven, and that our clients are bright*

sporty car with a powerful engine, stiff suspension, convertible top, and only enough room for two.

The logical solution is shown in the illustration. It's a combination vehicle with a little bit of what each driver wants: a convertible van with room for kids and lumber. What a goofy, impossible car! Even if it could be built, no one would want it. The correct solution is to build a minivan for Mom, a pickup truck for Joe, and a sports car for Seth.



Making three different products in software is a lot easier than making them in steel. Whereas there really must be three vehicles, one software product can usually be configured to behave like three different ones (with the caveat that the job of configuring it must not be dumped in the user's lap).

Every time you extend the functionality to include another constituency, you put another speed bump of features and controls across every other user's road. You will find that the facilities that please some users will interfere with the enjoyment and satisfaction of others. Trying to please too many different points of view can kill an otherwise good product. However, when you narrow the design target to a single persona, nothing stands between that persona and complete happiness.

Robert Lutz, the chairman of Chrysler, says that 80% of people in focus groups hated the new Dodge Ram pickup. He went ahead with production and made it into a best-seller because the other 20% *loved* it. Having people love your product, even if it is only a minority, is how you succeed.

The broader a target you aim for, the more certainty you have of missing the bull's-eye. If you want to achieve a product-satisfaction level of 50%, you cannot do it by making a large population 50% happy with your product. You can only accomplish it by singling out 50% of the people and striving to make them 100% happy. It goes further than that. You can create an even bigger success by targeting 10% of your market and working to make them 100% *ecstatic*. It might seem counterintuitive, but designing for a *single user* is the most effective way to satisfy a broad population.

The Roll-Aboard Suitcase and Sticky Notes

The roll-aboard suitcase is a good example of how powerful designing for one person can be. This small suitcase with the built-in wheels and retractable handle revolutionized the entire luggage industry, yet it wasn't designed for the general public. It was originally designed just for airline flight crews, a very narrowly defined target user. However, the product's design purity pleased this group enormously. The rest of the traveling public soon saw that it solved their luggage problem, too. Carrying it through crowded airports was as easy as maneuvering it down airliner aisles or stowing it aboard planes.

After the roll-aboard succeeded in its target segment, it was launched into other markets. Now you can buy double-sized roll-aboards, designer roll-aboards, armored-equipment roll-aboards, and kids' roll-aboards. Today, purchasing luggage without built-in wheels and a retractable handle is difficult.

As another example, an adhesive engineer at 3M named Art Fry used his own very specific requirements as a basis for creating what is arguably the most widely used and widely appreciated office tool. When he sang in the church choir, paper bookmarks always fell out of his hymnal, making him lose his place. Unwilling to damage the church's property with sticky tape, he looked for a better solution. He remembered an adhesive that he had worked on a few years earlier that was discarded because it didn't stick well enough. He used that failed adhesive to coat some small squares of yellow paper for bookmarks. This is how the 3M Post-it Note was born.

Happy users make remarkably effective and valuable assets. By narrowing your focus, you can generate fanatical customer loyalty in your target market. As discussed in Chapter 5, "Customer Disloyalty," customer loyalty can sustain you in difficult times. Not only will loyal users climb mountains and wade rivers to purchase and use your product, but they are the most powerful marketing tool known. Loyal users will personally recommend you to their friends. After you get buzz going about your product, you can build on it and extend your product into other segments of the market.

The Elastic User

Although satisfying the user is our goal, the term "user" causes trouble. Its imprecision makes it as unusable as a chainsaw is for removing someone's appendix. We need a more precise design tool.

Whenever I hear the phrase "the user," it sounds to me like "the elastic user." The elastic user must bend and stretch and adapt to the needs of the moment. However, our goal is to design *software* that will bend and stretch and adapt to the user's needs. Programmers have written countless programs for this mythical elastic consumer, but he simply doesn't exist. When the programmer finds it convenient to dump the user into the Windows file system to find the information she needs, he defines the elastic user as an accommodating, computer-literate power user. Other times, when the programmer finds it convenient to step the user through a difficult process with a mindless wizard, he defines the elastic user as an obliging, naïve, first-time user. Designing for the elastic user gives the developer license to code as he pleases while paying lip service to "the user." *Real* users are not elastic.

Programmers have an expressive taxonomy for describing the construction of software. Good programmers don't toss around gross generalizations about different computers and systems. A programmer would never say, "This will run well on a computer." Which computer? Which model? What operating system? What peripherals? Similarly, designers must never be so vague as to say their program "is designed for the user," or that "it will be user friendly." If you hear someone speaking like that, it is likely a way to justify the imposition of his own self-interest instead.

In our design process, we never refer to "the user." Instead, we refer to a very specific individual: a persona.

Be Specific

The more specific we make our personas, the more effective they are as design tools. That's because personas lose elasticity as they become specific. For example, we don't just say that Emilee uses business software. We say that Emilee uses WordPerfect version 5.1 to write letters to Gramma. We don't just let Emilee drive to work. We give her a dark-blue 1991 Toyota Camry, with a gray plastic kid's seat strapped into the back and an ugly scrape on the rear bumper. We don't just let Emilee go to work. We give her a job as a new-accounts clerk in a beige cubicle at Global Airways in Memphis, Tennessee. This distinctive specificity is very powerful as a design and communications tool. Consequently, all of our personas are articulated with singular detail and precision.

As we isolate Emilee with specific, idiosyncratic detail, a remarkable thing happens: She becomes a real person in the minds of the designers and programmers. We can refer to her by name, and she assumes a tangible solidity that puts all of our design assumptions in perspective. As she loses her elasticity, we can identify her skills, her motivations, and what she wants to achieve. Armed with this knowledge, we can then examine her in light of the software's subject area to see whether she is really an archetypal user. After a designer has some experience, he can usually synthesize a valid persona on the first try.

Giving the persona a name is one of the most important parts of successfully defining one. *A persona without a name is simply not useful.* Without a name, a persona will never be a concrete individual in anyone's mind.

All things being equal, I will use people of different races, genders, nationalities, and colors as personas. However, I try not to play against type because this can confuse everyone. Stereotypical personas are more effective if the stereotyping lends more credence to the persona. My goal here is not to be politically correct but to get everyone to believe that my personas are real. If my persona is a nurse, I will use a woman rather than a man, not because there are no male nurses, but because the overwhelming majority of nurses are female. If the user is a computer technician, our persona will be Nick, a pimply faced 23-year-old former member of the high-school audio-visual club, rather than Hellene, a statuesque, 5-foot-11-inch beauty who went to Beverly Hills High. I am shooting for believability, not diversity.

To make each persona more real to everyone involved in the product creation, I like to put faces to the names and give each persona an image. I usually purchase, for a small fee, faces from stock photo libraries on the Web. Occasionally, I've used sketched caricatures. You can cut them out of magazines if you want.

A fully realized, thoroughly defined user persona is a powerful tool. Until the user is precisely defined, the programmer can always imagine himself as the user or allow the user to become elastic. A completely defined user persona is key to the suppression of any tendency for the developer to usurp or distort the user persona's role. Long before a single line of code is written, a well-defined user persona becomes a remarkably effective tool for interaction design.

Hypothetical

It is important not to confuse a precise user taxonomy with a real person. Real people are of great interest as raw data, but they are frequently useless—and often detrimental—to the design process. A fine wine helps a successful dinner; raw Cabernet Sauvignon grapes—tiny, tough-skinned, and seed-filled—would ruin it. Many scientists, with a reverence for the empirical, confuse real users with imaginary—but more valuable—design personas.

The other major problem with real users is that, being real, they have funny quirks and behavioral anomalies that interfere with the design process. These idiosyncrasies are not extensible across a population. Just because one user has a distaste for direct manipulation doesn't mean that all—or even a plurality of—users do. The same works in reverse, too. Our real user might be fully capable of getting over some cognitive bump in the interaction road, whereas the majority of other users cannot. The temptation to attribute such capabilities to all users because one very real human exhibits them is strong but must be avoided.

In particular, we see this from company presidents. For example, one president we have worked with hates typing and wants to do all of his work without a keyboard. He has issued a directive that all of his company's software will be controlled only from the mouse. It is reasonable to want to use just the mouse to control the software, but it is not reasonable to shut out all those users who are more comfortable with the keyboard. The president is not a very representative persona.

Precision, Not Accuracy

As a design tool, it is more important that a persona be precise than accurate. That is, it is more important to define the persona in great and specific detail than that the persona be the precisely correct one. This truth is surprising because it is the antithesis of the *goal* of interaction design, in which accuracy is always more important than precision. The end result is to have a program that does the right thing, and we are willing to accept some friction in the system to obtain it.

In mechanical devices, moving linkages must be without slack. That is, a piston must move with minimal tolerances in its cylinder. If there were play in the linkage, the piston would quickly slap itself into self-destruction. It matters less that

the piston is too short or too long for the cylinder than that it fits without looseness. The same is true of personas. It matters more that the persona is expressed with sufficient precision that it cannot wiggle under the pressure of development than it does that it be the right one.

For example, if we were designing the roll-aboard suitcase, for our persona we could use Gerd, a senior captain flying 747s from Vancouver to Frankfurt for Lufthansa.

On the other hand, we can't extend our persona to include *any* commercial flyer. Sonia, for example, attends classes at Embry-Riddle Aeronautical University in Daytona Beach and will be a professional pilot when she graduates. She flies every day, but only in small, single-engine propeller planes, and never stays overnight away from home. From a luggage point of view, Sonia is an edge-case pilot. As soon as you blur the definition of Gerd to include Sonia, you make him approximate instead of exact. You get into endless, unproductive discussions about whether Sonia is or is not an airline pilot and what special features her baggage needs.

On the other hand, we could certainly design a roll-aboard by using Francine, a newly minted flight attendant on Reno Air, as a persona. She flies the length of California three times a day, serving drinks and handing out peanuts. Gerd and Francine are dramatically different personas, but their suitcase goals and needs are equivalent.

Programmers live and die by edge cases, and they will bring that awareness to the persona-selection process. They will argue that Sonia has a valid claim on persona-hood because she occupies a pilot seat. But whereas programming is defined by cases at the edge of the paradigm, design is defined at the center. If there is any doubt at all about a persona being very close to the center, that persona should be shunted out of consideration.

In the interest of being precise in the definition of personas, averages have to be ruled out. An average user is never actually average. The average person in my community has 2.3 children, but not a single person in my community actually has 2.3 children. A more useful representative would be Samuel, who has 2 children, or Wells, who has 3. Samuel is useful because he is a person. Yes, he is hypothetical, but he is specific. Our parent of 2.3 children cannot possibly be specific, because if he were, he wouldn't have that impossible average.

Average personas drain away the advantages of the specificity of precise personas. The great power of personas is their precision and specificity. To deal in aggregates saps that power.

Personas are the single most powerful design tool that we use. They are the foundation for all subsequent Goal-Directed design. Personas allow us to see the

scope and nature of the design problem. They make it clear exactly what the user's goals are, so we can see what the product must do—and can get away with *not* doing. The precisely defined persona tells us exactly what the user's level of computer skill will be, so we don't get lost in wondering whether to design for amateurs or experts.

The personas we invent are unique for each project. Occasionally, we can borrow from previous projects, but because precision is the vital key, it is rare to find two personas exactly alike.

A Realistic Look at Skill Levels

One of the really valuable contributions of personas is that they give discussions of skill levels a refreshing breath of realism. The scope of variation of users' skill levels is huge, and personas make that fact easy to see. The widely held, more-traditional model of user skill levels was shown as the euphemism pyramid in Chapter 2, "Cognitive Friction." At the top of the pyramid are "power users," assumed to be perfectly knowledgeable about computers, but lacking the training to program. The central trapezoid is "computer-literate users," who are imagined to have a fundamental understanding of how computers work, but who don't avail themselves of all its coolest features. "Naïve users" fill the bottom of the pyramid, and they are estimated to be as dumb as a brick and completely clue free.

Here are some examples of personas that shatter the pyramid's false assumptions:

> Rupak works as a network installer in Los Angeles. He works with computers all day every day and is very adept at getting them to function, but he doesn't really understand how they work. He survives through his reservoir of superstition and lore, his capacity for rote learning, and his endless patience.
>
> Shannon is an accountant at a health spa in Tempe, Arizona. She is clueless about the Web, email, networks, the file system, and most everything else about computers, but she is an astonishing whiz with the Microsoft Excel spreadsheet program. She can whip out a new spreadsheet—complete with charts and graphs—that shows sales trends in no time at all.
>
> Dexter is the vice president of business development at Steinhammer Video Productions in Hollywood. Dexter has a pager, two cell phones, a pocket computer, and a wireless modem stashed in the pockets of his double-breasted suit as he walks between sound stages. He is a master of technology, and he can solve any problem. His colleagues are always

calling him over to help find lost files for them, but he is really too busy for those time-wasting exercises. Clint is holding on line three!

Roberto is a telemarketing representative for J. P. Stone, the mail-order merchant of rugged outdoor clothing. He sits in a carrel in a suburb of Madison, Wisconsin, wearing a telephone headset and using a PC to process phoned-in orders. Roberto doesn't know a thing about high technology or computers, but he is a steady, conscientious worker and has a wonderful ability to follow complex procedures without difficulty. After a few days of training, he has become one of J. P. Stone's most productive and efficient reps. He says, "I *like* the computer!"

Interestingly, neither Rupak, Shannon, Dexter, nor Roberto comes close to fitting into any of the slices of the pyramid. Even aside from its oppressive stereotyping power, the pyramid is remarkably unrepresentative of the population. Oversimplified models of markets don't help with design problems.

Personas End Feature Debates

Surprisingly, another extremely important contribution of personas is their value as a communications tool. The cast of characters becomes a design taxonomy with great power to explain our design decisions. Even more, they become a spotlight, showing the programmers, marketers, and managers that our design decisions are obviously correct.

It is vitally important that everyone on the design team not only become familiar with the cast of characters, but that each persona become like a real person—like a fellow member of the development team. Programmers—with their mathematical nature—have a natural reluctance to think about specific user cases, preferring instead to think about general cases. This spills over into their thinking about users, and they are always imagining users in the aggregate, the average, or the generic. They much prefer to speak of "the user" than of Judy, Crandall, Luis, Estelle, Rajiv, and Fran.

Before personas are put to use, a typical conversation between a programmer and a manager engaged in interaction design would go something like this:

Programmer: "What if the user wants to print this out?"

Manager: "I don't think we really need to add printing in version one."

Programmer: "But someone might want to print it."

Manager: "Well, yes, but can't we delay putting printing in?"

The manager cannot win this discussion because she has not advanced an argument with the force of reason. Regardless of its truth, it is stated merely as her

amorphous desire to do things differently, and the programmer's logic of what "might" happen is irresistible.

After the cast of characters is developed, we have our taxonomy for expressing precisely who needs what in the program. But programmers are hard to move, and a typical discussion with a client programmer early in the relationship goes like this:

Programmer: "What if the user wants to print this out?"

Interaction designer: "Rosemary isn't interested in printing things out."

Programmer: "But someone might want to print it."

Interaction designer: "But we are designing for Rosemary, not for 'someone.'"

At this point, we are at a standoff. The programmer is still using the term "user" and is still stuck in the world of possibility thinking. However, our invocation of the persona Rosemary is not an amorphous, unformed desire. Instead, it is a specific person with a demonstrated skill set and objectives. We finally have an argument that is compelling.

However, because programmers have possession of the code, they can—and will—still do what they want, regardless of the strength of our arguments. The key to success is getting the programming staff to buy into the existence and reality of the cast of characters. Every one of our designers resolutely insists on expressing all design issues in terms of named personas. We *never* fall back into the "user" construct. We never let the programmer—or anyone else—get away with assertions about "the user." After a while, this consistency pays off, and the programmers begin to adopt personas and refer to them by name. Although this seems like a subtle change, when the programmers begin to speak of personas by name of their own volition, it is really a dramatic, watershed event that changes the nature of the collaboration between designers and developers.

This watershed occurs in every one of our successful design projects. When it happens, the entire process shifts into high gear. The conversations now sound more like this:

Enlightened programmer: "Would Rosemary want to print this out?"

Happy interaction designer: "No. Although Jacob will want some printed reports on a quarterly basis."

Enlightened programmer: "Well, if they are so rarely needed, we should save ourselves time and effort by not writing a fancy, proprietary report-writing feature, but instead license a commercially available tool."

Happy manager: "And that shaves two weeks off of the shipping schedule!"

I have seen dramatic changes come over our client companies after the watershed. Before, they were stuck in endless feature wrangling, and issues once thought resolved would reappear for further discussion every couple of weeks. Afterwards, design issues are raised, answered, and put away once and for all.

Some of our client companies have printed T-shirts with a picture of an important persona on it for each of the developers. We have had other clients print posters with personas to put on the walls of the programming shop. These efforts help to unite the programmers in a mutual understanding of their ultimate customer.

Both Designers and Programmers Need Personas

On the other hand, we have worked with companies where the programmers are simply too embarrassed to actually call users by their names, and they would not buy in to the idea of precise personas. They would continually backslide into "user-speak," and their products suffered enormously.

I know a programmer who simply doesn't understand how personas work. Under the pressure of arguments from my colleagues and myself, he has admitted that personas are important. However, he misses the central point of specificity, so he tends to use the term "persona" as a synonym for "user." He says, "We have to fulfill the needs of the personas." Although he uses the term, he rejects the specificity, which is the active ingredient, so he loses any value.

Another client gave us just a few days to make some recommendations. We created a persona named Edgar, who was not defined with much detail. We then entered some protracted discussions with the client on issues that extended beyond the original project scope. We quickly found that Edgar began to multiply. Different teams within the client adopted different Edgars, each with different qualities.

Marketing professionals will be instantly familiar with the process of persona development because it is very similar to what they do in the market-definition phase. The main difference between marketing personas and design personas is that the former are based on demographics and distribution channels, whereas the latter are based purely on users. The two are not the same and don't serve the same purpose. The marketing personas shed light on the sales process, whereas the design personas shed light on the development process.

As we begin to develop ideas for design solutions, we can constantly hold them up against our personas to see how well we have done. We become character actors, inhabiting the minds of our personas. This is easy to do because they are so narrowly defined. When you try on a persona and examine a product or a task, you can tell right away whether or not the design has succeeded in making that persona happy.

It's a User Persona, Not a Buyer Persona

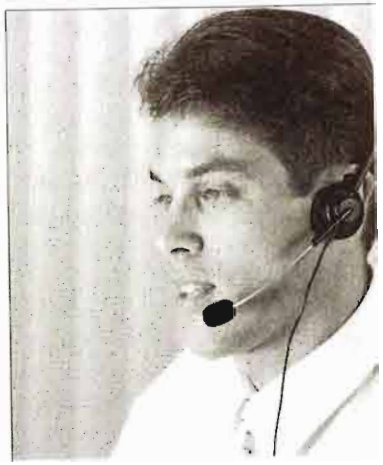
A frequent mistake is to design for someone who is close to the product but is not the actual user. Many products are designed for the writer who will review the product in consumer publications. In the information-technology business, the IT manager who purchases the product will rarely be the one to actually use it. Designing for the purchaser is a frequent mistake in the computer business.

While you cannot ignore the IT manager's needs, the IT manager will ultimately be happier if the product makes the *real* end user happy. After all, if the end user is happy and productive, it is a success for the IT manager. We have seen a recurring pattern in which clients ignore this advice and pander to these gatekeepers of technology. After these IT managers deploy the software to the real end users, they are besieged with complaints and discover that the users are disinclined to use the product that so beguiled the IT manager. They then turn on the software vendor, demanding that the interaction be made more satisfactory for the end user.

The Cast of Characters

We give every project its own cast of characters, which consists of anywhere from 3 to 12 unique personas. We don't design for all of them, but they are all useful for articulating the user population. Some are defined only to make it clear that we are *not* designing for them. In one project, for example, our project concerned a technical help-desk management system. We defined three people, two of them in-house help-desk technicians. Leo Pierce was a marketing assistant in the company's product division. He used a computer in his daily work and was occasionally a consumer of help-desk services. Alison Harding was a company technician whose job entailed going from office to office with her aluminum tool case, fixing technical problems for the likes of Leo. Ted van Buren was a help-desk representative who spent his day answering phone calls from people like Leo and dispatching Alison to Leo's office to fix his computer.

Our client, Remedy Inc, was revising its flagship product, Action Request System (ARS), and wanted to make it "easier to use." By developing these three personas (and a few others), we could clearly articulate what the goals of the project really were.



Ted was the main user of the current version of ARS, but he wasn't our primary persona. Although we would make operating the program easier for Ted, we would have failed in our job if that was all we accomplished. Instead, we were making the help-desk system directly accessible to Leo. Formerly, if Leo needed help, he had to telephone Ted, who would dispatch Alison. The full cast of characters articulated very clearly who the players were. This let us communicate to all the engineers that our goal could only be achieved if Leo, the low-tech marketing wonk, could use the ARS system on his own computer to summon technical help without Ted's intervention.

As soon as we could explain this situation in terms of personas, the team members immediately understood that they needed to deemphasize Ted and concentrate all of their efforts on Leo. Ted occupies a role we call a *negative persona.* His existence helps us to understand whom we are *not* designing for.

⌘

We know that we have isolated a persona when we have discovered a person whose goals are unique. It isn't necessary that all of the persona's goals be different, but that its set of objectives is clearly different from everyone else's. Raul, who assembles lawnmowers on an assembly line, has different goals from Cicely, his production supervisor. Cicely wants to improve overall productivity and avoid accidents. Raul wants to get a reasonable quantity of work done without making any embarrassing mistakes. Although they share practical goals, their motivations are quite different. Raul wants stability. Cicely wants a promotion. Clearly, their goals are sufficiently different to demand the establishment of two separate personas.

Primary Personas

Every cast of characters has at least one primary persona. The primary persona is the individual who is the main focus of the design. To be primary, a persona is someone who *must* be satisfied but who cannot be satisfied with an interface designed for any other persona. An interface always exists for a primary persona. In the Remedy ARS example, Leo Pierce was the primary persona.

Identifying the primary persona or personas is a vital step in the development of a cast of characters. In my experience, each primary persona requires a separate and unique interface. If we identify two primary personas, we will end up designing two interfaces. If we identify three primary personas, we will end up designing three interfaces. If we identify four primary personas, we know that we have a big problem.

If we find more than three primary personas, it means that our problem set is too large and that we are trying to accomplish too much at one time. We create personas to narrow down the spectrum of users for whom we design. It follows that

if the number of personas grows too much, we are defeating the purpose of creating personas in the first place.

The cast of characters is not just a convenient phrase; it becomes a physical—as well as logical—design tool. After winnowing down the population, we typically end up with anywhere from three to seven useful personas. We assemble all of them on a single sheet of paper containing their names, pictures, job descriptions, goals, and often telltale quotes. This one-page document becomes a ubiquitous part of our process. We print out copies of the cast of characters and distribute it at every meeting, whether or not the client is present. Every designer at all of our brainstorming meetings and all of our detailed design meetings has a cast-of-characters document in front of him at all times. When clients attend these meetings, extra copies are printed and presented to them. Every deliverable document that we create and give to our clients has a cast-of-characters page in it. Our goal is to make the personas unavoidable. They are so important that we cram them down everyone's throat.

It does no good to have good design and not express it in terms of the user personas. It is simply too easy to slip back into speaking about "the user" and lose the hard-won focus on specific archetypal users.

Case Study: Sony Trans Com's P@ssport

In 1997, Sony Trans Com approached us with a remarkable design problem. Sony Trans Com is Sony Corporation's Irvine, California, division, responsible for the design and manufacture of in-flight entertainment (IFE) systems. In-flight entertainment—movies, TV shows, and video games in commercial aircraft—is a large and lucrative business. Sony Trans Com had developed a new generation of technology that brought a new level of capability to airline passengers. The most impressive capability of the new system, called P@ssport, was true video-on-demand (VOD). VOD lets Tricia in seat 23A begin watching *When Harry Met Sally* 10 minutes after takeoff, and it lets Anna in seat 16C start the same movie 45 minutes later—and either passenger can pause or rewind the show without affecting the other.

P@ssport pushed the envelope of IFE well beyond the current technical state of the art. Each seat back contained a video screen and a Pentium computer running Windows 95. In the front of the plane was a powerful array of computers with copious memory for content. A fiber-optic cable connected each seat to the array, with connector boxes placed every few rows throughout the plane, making the system blindingly fast and breathtakingly powerful.

Sony had worked on this system for months before it asked us to help design the interaction. Although the engineers were making good progress, their designers were at an impasse. Just about anybody could occupy an airline seat, so they were trying to accommodate everyone from the total computer novice to the computer expert. They had no idea how to please all those constituencies. Neither did we, but we had our powerful design techniques, including personas, and were confident that we could solve the problem.

The Conventional Solution

Sony Trans Com had already designed and built a prototype of the P@ssport system with a conventional interface. It was very consistent with the program's internal structure—that is, it was very implementation model. Basically, it consisted of a deep hierarchical tree of screens through which the user had to navigate, making decisions at each screen. The evident shortcomings of this prototype are what prompted Sony to approach me.

Each screen represented another layer in the hierarchy, and it required six of them to examine each movie selection.

It was a classic example of what I call *uninformed consent*. At each step, the user is required to make a choice, the scope and consequences of which are unknown. At the first screen the user must choose an entertainment type: music, movies, games, shopping, and so on. Selecting "Video" makes all the other choices disappear, and the next screen demands that the user choose the category of film. The screens keep coming until, at the sixth level down, the user can see a brief preview of the movie and then choose to watch it or not. If she decides to pass, it's six clicks and six screens back up to the top, and then six clicks back down to view the next one. Whew!

Because P@ssport runs on a screen in each seat back, every user is within arm's length of the screen. It was instantly obvious that a touch screen was a great, natural solution, rather than using a handheld remote controller. But Sony rejected the idea anyway. Sony realized that—with six levels down and six levels up for each selection—it would take several dozen taps for the typical user to select entertainment. It realized that the person sitting in front would be enraged by all the tapping on the back of his or her head. In a classic example of what Po Bronson means when he says engineers will keep fixing what's not broken until it is broken, Sony discarded the touch-screen idea and reverted to a handheld controller tied to the seat with a short wire. It threw the baby out with the bathwater. The engineers regretted the decision but saw it as inevitable in light of their schedule constraints.

The six-screen interface was a classic example of implementation-model design, accurately reflecting the internal choices of the software. Each decision screen offered up very little context or supporting information, so the user never felt oriented, which made navigation a big problem. Each time the user drilled down into another layer, she lost the current context. After she committed to "Video" she was no longer able to select—or even see—the "Games" option. At each step of the way, the program was ignorant of the bigger picture, so it kept the user similarly ignorant. The user had to choose "Video" without knowing which or how many movies were available. She then had to choose a single category of movie, again without knowing what those categories meant. Is *True Lies* an adventure, a romance, or a comedy? When she is finally shown movie titles, she is bereft of information. Hmmm, *Eraser*, wasn't that an art film about a mild-mannered schoolteacher?

But even while still in the prototype stage, the interface had beautiful 3D graphics, very artistic icons, and a map-and-globe metaphoric theme—all the trappings of a good interface without the substance. It's what we call "painting the corpse."

Personas

As always, our design process began with a thorough investigation phase, consisting mostly of interviews, beginning inside Sony. We listened to most of the people working on the product, including the project manager, the development manager, a couple of the engineers, the product marketing manager, and the content manager. These interviews gave us a good idea of what Sony Trans Com wanted to accomplish with the product. It also gave us some historical perspective on past IFE business and technology. Armed with this knowledge, we shifted our interviewing process to the field. We listened to lots of airline personnel, particularly flight attendants from several airlines.

During the interview process, our design team kept inventing new personas. Every time a flight attendant would tell us a new story, we'd add another persona, until we had about 30. The more we listened, the more we learned, and eventually the similarities among the personas became apparent. As we found personas with common goals, we could collapse them into one. Eventually we narrowed the persona population down to 10; four passengers and six airline employees. As you might imagine, the airline employees with formally described jobs and responsibilities were pretty easy to understand and design for. The tough nut was the passenger persona. Each of the four passenger personas was an archetype in its own way, representing a broad segment of users, but you can't design an interface for four personas. You have to find the one common denominator. Here are the final four:

Chuck Burgermeister, business traveler. A 100,000-mile-club member who flew somewhere practically every week. Chuck's vast experience with flying meant that he had little tolerance for complex, time-consuming interfaces, or interfaces that condescend to novices.

Ethan Scott, 9-year-old boy. He was travelling unescorted for the first time. Ethan wanted to play games, games, and more games.

Marie Dubois, bilingual business traveler. English was her second language. She liked to browse the shopping, as well as the entertainment selections.

Clevis McCloud, crotchety septuagenarian. An aging but still spry Texan, slightly embarrassed about the touch of arthritis in his hands. He was the only one of the four passenger personas who didn't own a computer or know how to use one.

PASSENGERS

Clevis McCloud
Age: 65, World Odyssey Class

Marie Dubois
Age: 31, Odyssey Club Class

Chuck Burgermeister
Age: 54, Odyssey Gold Class

Ethan Scott
Age: 9, World Odyssey Class

ODYSSEY AIRLINES CREW

Brent Covington
Age: 37, Purser

Amanda Kent
Age: 28, Flight attendant

Jean-Paul Duroc
Age: 33, Interpreter

Molly Springer
Age: 41, Content renewal specialist

Mel "Hoppy" Hopper
Age: 51, Mechanic

James A. Tattersall
Age: 47, Pilot

Our interface had to satisfy Chuck, Ethan, Marie, and Clevis while not making any one of them *unhappy.* But that didn't mean that we had to make all four of them exceptionally happy. Ethan knows that wanting to play games, games, games is something special, so he doesn't mind pressing an extra few buttons to obtain exactly what he wants, as long as he *can* obtain it. Chuck knows that his vast experience has earned him some shortcuts, but he is willing to put a little extra effort into learning and remembering those special commands.

Clevis turned out to be our common denominator. Clevis didn't have or want a computer. His motto was, "You can't teach an old dog new tricks." He wasn't stupid or lazy, just not an apologist for the antics of high technology. We knew that

if we put a caption bar and close box on the screen, we would instantly lose Clevis. This meant that all computer-like interfaces were out of the question. We also knew that with his arthritis, no complex manipulation would be acceptable. He should be able to operate the system with the ball of his hand.

Any solution that focused on Chuck, Marie, or Ethan would be unacceptable to Clevis. Clevis would be scared and confused by Chuck's shortcuts and Marie's language-selection options. Ethan's twitch games would just get in Clevis's way. Yet a solution that made Clevis, the crotchety old Luddite, happy would be perfectly acceptable to Chuck, Ethan, and Marie, as long as their special needs were accommodated somewhere in the interface.

Chuck and Marie were old hands at flying, so they could find their way around any system, as long as it didn't involve a lot of time-consuming training screens for new users. We knew that if we made the system very simple and visual, it wouldn't involve a lot of interaction overhead, and Chuck and Marie wouldn't be offended. Ethan was easier, because we knew that he would quickly and aggressively explore the system to find out where everything was. As long as his games weren't hidden away somewhere, he'd be happy.

Throughout the entire design project, Clevis was our touchstone. His image became our battle standard. We knew that to make Clevis happy would mean that we would make any and every airplane customer happy. He was our primary persona, and we designed the system for him and him alone.

Designing for Clevis

Clevis had no experience with computers and no patience for the typical attitude of delayed gratification that most programs have. The solution to Clevis's navigation problem was simple: He could not and would not "navigate," so there could be only one screen. The solution to Clevis's reluctance to explore the interface meant that the product had to be very generous with information. We were parsimonious with choices but copious with information.

We turned the screen into a horizontally scrolling panoply of movie posters and album covers. We created a large, rotating knob—which we call a "data wheel"—that Clevis could spin like a station selector on a radio. This was an actual knob on the bezel of the screen, not an image of a knob drawn in pixels. As Clevis spins the data wheel, the posters scroll smoothly by, moving right if the knob is turned clockwise and left if it's turned counterclockwise.

Clevis views the posters going by, like strolling down Broadway peering into store windows. He never has to choose—or even think about—what category a movie is in. Because there was no tree of choices to traverse, we reinstated the touch screen without requiring woodpecker-like tapping. When Clevis sees a movie

poster that interests him, he taps it once and immediately sees and hears a brief preview, along with written reviews, cast, crew, and pricing information. Clevis can then tap to watch the movie or tap to resume his leisurely stroll down Movie Street.

The scrolling movie posters are arranged in a manner we call a *monocline grouping*—a single layer of information organized into groups. We often replace interface hierarchies with them. The top of your desk is likely organized in a monocline grouping, as are your bookshelves and your bedroom drawers. People, including Clevis, Marie, Ethan, and Chuck, very quickly and easily grasp monocline groupings. Monocline grouping changes the "category" of movie from a *necessary choice* that must be made in advance into a *helpful attribute*. Clevis can peruse movie posters and see what category any given movie belongs to without committing to it. It also neatly solves the problem of selecting which category something is in. The movie *True Lies*, for example, is an action film, a star vehicle, an adventure, an effects showcase, a romance, and a comedy. A hierarchy would force it into just one of those categories, but in a monocline grouping, it can easily have all of them as attributes.

After Clevis scrolls through the movie posters, the images seamlessly change, first to album covers and then to game posters. There are few enough selections that Clevis can merely spin the knob and bring everything into view. The knob is large enough, and its motion coarse enough, that even Clevis, with his large, hard, oilfield hands with their touch of arthritis can still rotate it with ease. A navigation bar across the bottom of the screen informs Clevis that there are several broad categories of entertainment, and a small indicator on the bar moves to point to his place in the spectrum of choices.

The Sony programmers fell into the three-number trap of 0, 1, and infinity. The P@ssport system can handle, in practical terms, about three dozen movies. From a programmer's point of view, 36—being greater than either 0 or 1—is the same as infinity, and the idea of presenting an *infinity* of movies seemed problematic, so they divided them up into categories. But Clevis enjoys scrolling through the three dozen choices. Even if there were a few hundred movies, he'd still enjoy the leisurely browsing, remembering movies he had seen and anticipating seeing those he hadn't.

A lot of the value of the solution was in the posters, which convey significant information about each movie: the stars, the plot, the attitude. The engineers saw this but were concerned that offering up movie posters would create extra work for the content providers. When we broached the idea to some movie vendors, they reacted in just the opposite way. They were ecstatic at having the opportunity to get their posters into the interface. After all, they had spent hundreds of thousands of dollars to have experts produce a poster that conveyed, most informatively and concisely, as much information as possible about a film and appealed to the broadest audience. Why not put that to use on the airplane? They considered it a great new opportunity to produce bitmaps for the product.

Although we designed the passenger interface for our one primary passenger persona, we made certain to provide for the needs of the secondary passenger personas. Chuck Burgermeister, the frequent flyer, will want some shortcuts, and those are built into the interface so that Clevis won't even notice them. If Chuck wants to move between categories of entertainment more quickly than the data

wheel allows, all he has to do is touch the navigation bar at the bottom of the screen. The program immediately jump-scrolls to that part of the monocline grouping without forcing Chuck to scroll there. Clevis never even needs to know about this ever-present idiom, yet it is very easy to discover and learn, and more-experienced travelers like Chuck and Marie will quickly pick up the trick, either from their own exploring or from watching others.

Unlike images on a screen, physical knobs and controls invite manipulation. When Clevis first sees the data wheel, he can easily intuit from its shape and orientation how to work it. Although Clevis cannot intuit its behavior, all it takes is a single, small turning, and its behavior and effects are instantly clear because an equal motion of the movie posters on the screen instantly echoes any motion of the wheel. More probably, Clevis will see some other passenger spin the wheel and see the image scroll in direct proportion. The one-to-one relationship between wheel and screen is instantly understood, and Clevis has learned how to use the system in an instant.

⌘

I've only described the interface we designed for Clevis McCloud, the passenger. We also designed two other large interfaces, one each for the other two primary personas: Amanda Kent, the flight attendant, and Mel "Hoppy" Hopper, the mechanic. Their goals are quite different from Clevis's.

After safety, Amanda must focus on ensuring that each passenger has the best experience possible. Her interface must provide control over all in-flight operations. For example, if Chuck in seat 24C wants to move because Clevis in 24B has fallen asleep and is snoring loudly, Amanda must be able to transfer Chuck's account and his half-viewed movie to 19D, the empty seat that he moves to.

Hoppy's main requirement is to assess rapidly the state of the system. He determines what is malfunctioning, how serious it is, and what he can do to fix it.

Both Amanda and Hoppy use the same screen at the flight attendant's station, but their interfaces are dramatically different because their goals are different.

⌘

If you want to design software-based products that make people happy, you have to know who those people are with some precision. That is the role that personas play. The next step is designing the product to be as powerful as possible, and for that, you need to know more about the user's goals.