popu-
ng the

of-use
ive the
We are
l up to
e with

and to
prepa-
d, Nick
lebeek-

nt. *IEEE*

Iry, CA:

national

CHAPTER Four

Designing the PalmPilot: A Conversation with Rob Haitani

ERIC BERGMAN
Sun Microsystems, Inc.

ROB HAITANI
Handspring, Inc.

FIGURE 4.1

Rob Haitani

BERGMAN: *You worked on the design of the original Pilot (see Figure 4.2) at Palm Computing. What was your role?*

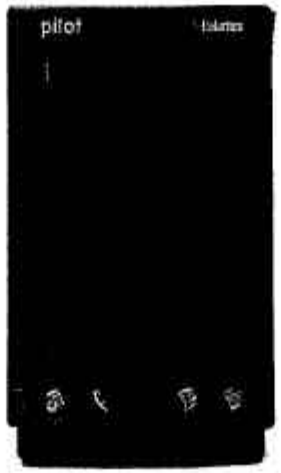
HAITANI: I was the product manager for the first-generation Pilot. Among other things, I was in charge of designing the user interface of the operating system and the applications. Our president, Donna Dubinsky, one morning asked me if I had ever "done this before." I wondered if she meant whether I had designed an operating system. I hadn't even managed a software project before coming to Palm, actually, so I simply said, "Uh, no."

BERGMAN: *Tell me a little bit about the history of the PalmPilot.*

HAITANI: Before the PalmPilot, we wrote the application software for a product called the Zoomer, which was similar to the Newton. It had a 320 x 240 pixel touch-sensitive display, a pen, and handwriting recognition. It was called "jacket pocket size," which was another way of saying that it was too big to fit in a

FIGURE

FIGURE 4.2



An early PalmPilot connected organizer.

shirt pocket. We were actually a software company, and our expectation was that these products would become smaller, cheaper, and faster over time, and that we would be the leading software developer for whatever product won out. Trouble was, we didn't anticipate that *none* of the products would win.

What was more discouraging was that the hardware vendors didn't understand the basic problem. They thought the first-generation products failed because they did not provide enough functionality. The second-generation products, much to our dismay, were even bigger, slower, and more expensive. One day at a board meeting, our founder, Jeff Hawkins, was lamenting the fact that the hardware companies didn't know how to build the right product. One of the board members turned around and asked him if Jeff knew what the "right product" was. Jeff answered yes. "Well, then why don't you build it?" he asked. And that's how the Pilot started.

Jeff believed we had to make the product considerably smaller then current PDAs. He carved up a piece of wood in his garage and said this is the size he wanted. He'd walk around with this block in his pocket to feel what it was like. I would print up some screenshots as we were developing UI, and he'd hold it and pretend he was entering things, and people thought he was weird. He'd be in a meeting furiously scribbling on this mockup, and people would say, "Uh, Jeff, that's a piece of wood."

BERGMAN: *How did he decide on that size for the piece of wood?*

HAITANI: He believed it had to fit in your pocket, and he did some stack-up analysis to study feasibility. The batteries and the LCD will be this big, so it will be about this thick. That's where we started. Then the question was how to get everything to fit. Making the smallest possible product eliminates much of your flexibility. It's not too much of an exaggeration to say that the buttons are where they are because that's the only place they'd fit. Although that begs the question of why we had buttons rather than on-screen software buttons or buttons printed on the LCD. This approach was counter to the current trends in the industry. The whole point of a computer is that you don't have to have mechanical buttons. You can just turn it on, and then you can put buttons on the screen wherever you want.

But we decided that instant access to your data meant that when you want to see your schedule, you don't want two presses. You don't want to have to pull your pen out. You want one-touch access to your schedule. The only way you can do that is if the application button doubles as a power button.

BERGMAN: *Was there argument about these non-PC design elements during development?*

HAITANI: It was a little controversial, but maybe not as much as you might think, since we were so focused on making it work rather than making a little PC. Another element that would have been more controversial if you took a PC approach was behavior when switching applications. When you press the Date Book button, it only shows today's schedule, and from a PC perspective, that's illogical. On a PC, when you toggle between apps, you want to go back to where you were. It's a no-brainer. In our devices, however, you may be checking a day, then turn the device off. Next time you come back to your calendar it might be hours or days later. You could be trying to take a quick glance at your schedule and not realize you were looking at tomorrow's schedule. Your eye doesn't necessarily go to read the little date at the top of the screen. You're much more likely to want to see your schedule for today than you are to see the last day you happened to have been looking at. Again, from a PC perspective that's not very logical, but that's what you want.

BERGMAN: *Do the right thing?*

HAITANI: Yes, just do the right thing. And the things you do on a PC are different than what you do on a handheld.

BERGMAN: *How did you arrive at the particular display size that's used now?*

HAITANI:

BERGMAN:

HAITANI:

BERGMAN:

HAITANI:

HAITANI: It was fairly arbitrary. There were no standard handheld LCD sizes. Jeff was looking for something equivalent to Game Boy screens, so we went for a 160 × 160 resolution. Actually we started at 160 × 128. Jeff came to me one day and asked if I thought we could design an interface that will fit into that screen size. I said with conviction, "Oh absolutely, no problem." Then when he left, I thought, "Oh boy, this is a big problem." When we changed to 160 × 160, I was ecstatic.

BERGMAN: *So you didn't actually know whether you could or couldn't?*

HAITANI: Well, at the time Jeff just said this is the screen size, and everyone was okay with that. But as the person who had to implement it, I was the only person who actually thought, "Is it even feasible?" Everyone else assumed that it was. But I felt it was a great challenge and I wanted to take it on. It reminded me of how Japanese companies have the concept of hardware miniaturization in making a Walkman or a camcorder, and I wanted to apply that same concept to software—in other words, "software miniaturization." Can we take the same amount of information and display it in fewer and fewer pixels?

BERGMAN: *So where did you start?*

HAITANI: Well, I started with the date book. I wanted to display a full day, eight to five, six preferably. And that was our starting point. To fit that in there you very quickly find that every pixel counts and that there's just no room. Any embellishment adds more pixels. Your creativity is very constrained and you can't add a lot of flourishes or fancy graphics. For example, we even eliminated drawing a single line around the main application screen as a border, like you would find on typical PC applications. If you have a border, that means not only do you have to allocate one pixel for the border itself, but you need a margin of three or four pixels between the border and the text. So that's five pixels on four sides. When you only have 160 pixels, that's a huge percentage of your real estate, something like 12%! We realized that because LCDs have a nonactive area, however, there's a natural margin between the LCD pixels and the bezel of the display, so we could gain that space back and draw pixels right to the end of the active area (see Figure 4.3). Another seemingly small example is font size. If the standard average font width is five pixels instead of six, then you fit an additional 16% text on a screen. It doesn't sound like a lot, but these things start to add up—it's like hardware miniaturization. A half-millimeter smaller component is not a big deal, but the cumulative effect makes a big impact.

FIGURE 4.3

Every pixel counts.....

BERGMAN: *So you fought for every pixel....*

HAITANI: Absolutely. For example, we quickly jettisoned any 3D treatment. Any shadow takes up at least one more pixel on each side as compared to a single black line. I spent a lot of time agonizing over font height—we ultimately shaved a pixel off of accented capital letters. They could have looked better, but we felt it was worth getting the extra text (and you know, the French don't use capital accented letters . . .). Most screens get about 11 lines on the page—that extra pixel bought us an extra hour in the day view of the calendar! Or to put it another way, we eliminated some percentage of the times a customer had to scroll to see all the events in their day.

BERGMAN: *Take me back a step to give a sense of designing the product as a whole. How did you decide what was going to be in the Pilot? What wasn't? How did it come to have the set of functions it had and the hardware design it had?*

HAITANI: I think we learned a lot from the Zoomer. On the Zoomer, our philosophy was that we should put as many applications as possible in to make as many people happy as possible. After it shipped, though, we did some user research that made us question that decision. We found that there were a few applications that people used extensively, and usage very quickly dropped off after that. So we said why burden the product with all this extra functionality if people aren't going to use it? We very quickly decided to focus on the basic PIM [personal information management] software: date book, address book, "to do" list, memo pad, and calculator. Then we designed the hardware to support that basic functionality.

BERGMAN: *Were there other functions under consideration that ultimately didn't make it in the final product?*

HAITANI: Yes, actually there were. The conventional wisdom in the handheld market at the time was that the first-generation products didn't succeed because they didn't have enough functionality. If you're going to spend $700 or $800 on a handheld device, then it's really got to do something for you. Specifically at the time the "killer app" was wireless two-way email. That was supposed to be the key. If only PDAs did that, then they would be successful. We held focus groups and went out of Silicon Valley to talk to people. We pitched this email concept, and I remember very clearly one woman who just had a blank look, and she said, "I get three emails a day. I check my mail once every morning. Why do I need access in a handheld device?" When you're in Silicon Valley the tech frenzy starts feeding on itself, and you end up losing context of what real people do with real products.

BERGMAN: *So, when you went out to find out about what people would want, what was your approach or process?*

HAITANI: First, we showed people a mockup of the device, just the little organizer, and asked them what they thought of it. They were mildly positive, and thought it was okay. But unsolicited they said, you know, unless it's hooked up to my PC it's not very interesting, or I really need this linked to my PC. So then we showed them a simulation of synchronization with HotSync and the cradle, which at the time was a revolutionary concept, and they went totally nuts. They went gaga over it. Then we were sitting behind the one-way mirror thinking that when we show them the email, that's going to be a slam dunk. But it died, like a lead balloon. And we thought, maybe we should just focus on the basics. What they were telling us is, "I don't need the bells and whistles. I just want to organize my phone numbers and my schedule. Give me something that does that and then hooks up with a PC very elegantly, and that's great."

BERGMAN: *Was the HotSync part of the product from early on, or was that something suggested by what you found from people?*

HAITANI: No, that was originally a core part of the concept, and was another example of what we learned from the first generation of PDAs. If you remember back to the original vision of PDAs, they were designed to free you from your PC—you wouldn't have to be connected to your desktop. Then they hedged a bit, saying you might want to back up some data once in a while, and provided data backup applications. But since they assumed it was a low priority, those applications were slow and crashed a lot. Then our user research on the Zoomer product told us that 90-odd percent of our customers owned PCs, and we came to the conclusion that you *don't* want to be freed from your computer. You still have important information that's on your PC and you need access to that information. We decided that we needed to design that with that assumption in mind. We had to make the data link to the PC a no-brainer to use. Other people had done it before, but it was always very cumbersome and half the time it didn't work. We said from the beginning we need to make it very simple, so you could press one button and it synchronizes to the PC. It sounds obvious in hindsight, but was another revolutionary aspect of the product.

BERGMAN: *What were the criteria you used to evaluate the success of your designs?*

HAITANI: At the time we viewed the world as consisting of two main types of competitors. On the one hand, there was the electronic organizer market, which was already very well established with players like Sharp and Casio who sold millions of these things. Some of these products were very small, fast, and inexpensive, but had tiny keyboards and were too complicated to figure out how to use. Sometimes for grins Jeff would give someone a product and challenge them to figure out how to set a meeting or enter a phone number within 10 minutes. At the other end of the spectrum, there were PDAs with better interfaces such as the Newton, but they were big, slow, bulky, and expensive. We refused to believe that this was an intractable problem, that there wasn't a better middle ground.

BERGMAN: *Can you elaborate on what ways devices can be too slow or complicated?*

HAITANI The big problem with our original Zoomer product was that it had a very slow processor, and we found that it was excruciatingly painful to use because it took so long to get simple things done. When I first came on board with Palm, my first job was to spec the second generation of the Zoomer software. It was a software update, so there was nothing we could do about the processor. So we asked, what can we do about performance? I did a little bit of analysis and came to the conclusion that the processor wasn't entirely to blame. Our interface was inefficiently designed and required extra steps to

accomplish simple tasks. If you're just setting up an appointment or looking up a phone number, the more steps required compounds the fact that you have a slow processor because every step takes a couple seconds. And at that time, it would take literally 20 or 30 seconds to change the time of an event or enter a phone number. I spent a lot of time stripping that down and asking how could we reduce the number of steps to do common things. Instead of eight taps could we reduce it to five or three or two? We spent a lot of time optimizing navigation, etc. Got into a lot of arguments also.

GMAN: *Arguments?*

BERGMAN:

ITANI: Well, someone would say, "That's just one more tap," or "That would only take another second." I would respond that it is analogous to the way you organize your desk in your office, your physical desk, in that you have some things on top of your desk and you have some things in drawers or file cabinets. Why is that? Well, if you look at the things on top of your desk, those are the things you use very frequently and they need to be easily within your grasp; whereas things in your drawer you don't use as frequently. So I would say, Imagine taking something you use all the time like your mouse or the phone and put it in a drawer. It's just one extra step to pull it out. It just takes a second. But if you use it that frequently, the cumulative effect of that one extra step is excruciatingly annoying. On the other hand, one more tap does not matter for features you use infrequently. For example, if you have a weekly meeting at four o'clock, which we call a repeating event, you may find out that the regular time has changed. How many times a day do you do that? Maybe once every couple of weeks. Therefore *in that case* it's not a big deal to require the customer to take a couple extra taps to reach that dialog. They don't get annoyed; they don't even notice.

HAITANI:

GMAN: *I understand one measure you used to evaluate your designs was something called the "phone test." What was the phone test, and how did it come about?*

BERGMAN:

ITANI: In the days of early PDAs, you'd be on the phone with someone and they'd ask, "Can we meet next Tuesday?" Then you'd be tapping on your screen and watching wait cursors, and there'd be this dead air you'd have to fill with small talk, which was very embarrassing. So we said that's one thing that we have to fix. It's got to be instantaneous. It's got to be fast. I came up with an informal concept called the "phone test," which was similar to the Turing AI test [can you tell if the writer on the other side of a computer connection is a human or not].

HAITANI:

Our challenge was to see if we could design a product fast enough that the person on the other end of the phone doesn't realize you're using an

electronic device. The experience we had gained designing apps to minimize the number of steps came in very handy. When these lessons were applied to the Pilot, it was like baseball players warming up with extra bats. When you take the weights away you're really swinging hard. We designed Pilot to be very fast, particularly the things you want to do very frequently and have instant access to.

I actually remember the first day we had the applications running. We fired up the boards and we had a simulated phone test. I said, "Ed, what are you doing next Tuesday? How about 12:00 P.M.? Do you have Ron's phone number? 555-1212." It was kind of a joke, but we literally validated the phone test, which was fun.

BERGMAN: *You spoke earlier about the frequency of tasks and the number of taps. In your design approach, what was the relationship between task frequency and number of taps required for users to perform a task?*

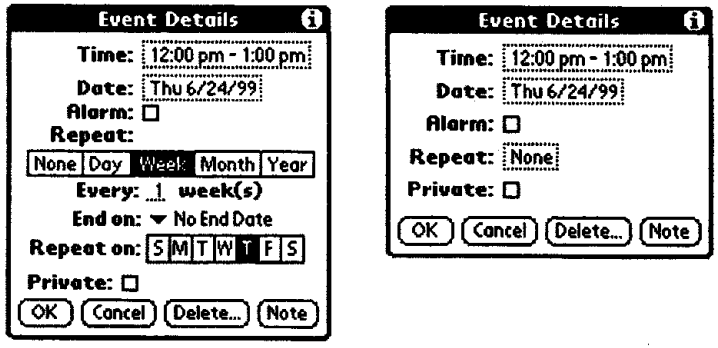
HAITANI: In designing for small screens, we very quickly ran into two conflicting design goals. On one hand, in order to be very fast and have access to features, you want them right on the screen so you can have just one tap to get to everything. On this end of the spectrum the ideal application is to have all the buttons on the screen. On the other hand, since we have a small screen, we don't have a lot of real estate and we want to make it easy to use. You want to make it very simple and you don't want clutter. So the other end of the spectrum is that you want an application that has as few things on the screen as possible. Obviously these goals conflict.

We struggled with this until I came to the conclusion that there's a way out. The key is that most people only use a small percentage of the features in an app. If you drew a chart of features sorted by frequency of use, what you find is there are a few features that people use all the time, and then it very rapidly drops off. Think of your word processor. Cut, copy, paste you use over and over and over again. But things like multiple column layout and drop letters most people use once in a blue moon. This led me to conclude that the features that you use very frequently need to be right up front, but features that you use infrequently, you can bury. That was the breakthrough. That allowed us to design screens that were simple enough to be easy to use, yet providing instant access to 80% of the features you use. [See Figure 4.4 for an example of this approach.]

BERGMAN: *That sounds great in theory, but how did it work in practice?*

HAITANI: From a design perspective, it worked remarkably well, providing you had a good enough understanding of your customer to know what features were really important enough to have on the screen.

URE **4.4**



Datebook dialog before and after. Repeat appointment features were moved into a dialog (not shown) that comes up only if the user chooses repeat appointments.

But that approach tends to generate resistance from engineers. It was difficult conceptually to grasp the concept from an execution perspective because it leads to inconsistent and illogical results. When you design something, you assume things that are equivalent should be placed together. For example "new record" and "delete record" commands, well, those should be next to each other. But I would argue that you're much more likely to add a new phone number than you are to eliminate one. And therefore the New button should be right on the screen, but the Delete button you can safely put behind a menu. We had a lot of heated arguments about that.

Again, I'd point to how your desk is organized. When you think about it, your stapler and your staple remover are "architecturally equivalent" and therefore theoretically should be in the same place. Either both of them are in your drawer, or both of them are on top of your desk. In my case, though, I have my stapler on top of my desk and my staple remover in the drawer. That's because I staple papers much more frequently than I remove staples from papers. If you explain it that way, people tend to get it.

SMAN: *So instead of enforcing consistency, would it be fair to say that the design goal was to maximize predictability?*

ITANI: Yes, exactly. And there's no real scientific method about it. I get great satisfaction out of hearing people say things like, "Gee, the PalmPilot just does what I want to do," and they don't necessarily understand why. They don't care why, but yet it just *works*.

BERGMAN: *Were there any cases where due to testing or feedback you found a particular context where this predictability approach didn't work? Where perhaps you said, we're going to have to bury this function because people hardly ever do it, and then it turned out that finding that function was too difficult? Or cases where you said we really need this function up front, but it turned out to be confusing in that location?*

HAITANI: One thing I think was very important was that I did a great deal of user testing and simulation using HyperCard. It was a very valuable exercise, and we actually did a whole pass at the user interface before any of the code was even written. Many people make the mistake of waiting until alpha to user test, when a functional prototype is available. But by that time it's already architected in a certain way, and to change it introduces a lot of risk, and you end up with buggy products that don't seem to hold together as well. On the other hand, if you decide right up front to spend the time, it's always worth that extra investment.

For example, in the date book we have a very odd design where you can tap right on a line and start to write down a meeting. It introduces a whole host of problems. What if you're in the middle of entering text, and tap somewhere else on the screen? What happens if you tap a command button or pick list when the focus is on the text? It introduced a lot of issues. If your objective is to make a clean design, it's not a really good idea. The more logical PC approach is to have a dialog displayed to edit the settings like alarms. That way you can explicitly confirm or cancel the editing session. I fought against that very hard because I felt that most of the time you're setting up a meeting at, say, one o'clock, and you just want to write it in. When I wasn't able to convince people, I'd say let's defer it and test it. We'd user-test it and I was either right or not—and if I wasn't right, and we built a better product, that's fine!

That approach generated a problem we considered addressing with modality. Since we weren't displaying a dialog, there was not enough space on the screen to put all the buttons we wanted. When you have a record selected, you want cut, copy, paste, etc. And at one point we had an idea that when you tap on a line, it puts you in editing mode and displays a different set of buttons. It made sense since you couldn't edit a record if you hadn't selected it, so those buttons weren't serving any purpose if nothing was selected. As a result, it gave us a lot more flexibility to display more buttons. But when we simulated this design and tested it in HyperCard, people were confused by the buttons that appeared and disappeared.

That painted us into a corner. We can only put four buttons on the screen, so what are we going to do? We could try cramming smaller buttons on the screen, but as I said, we wanted to minimize clutter. We were forced down a path that led ultimately to our design principles. To be honest, it wasn't a case where philosophically we grasped a breakthrough in interaction design. It was more an end result of our pragmatic design approach, starting with the fact that we could only fit four buttons on the screen. Instead of trying to apply a fancy UI model to address the problem, we simply asked the question, "Which four buttons should we put on the screen?" In other words, forget rules and theory, just do the right thing. Over time, however, I started recognizing patterns in our implementation. Our UI seems to be holding together—hmm, I wonder why. Now that you think about it, there's a pattern here. Eventually I was able to articulate what these patterns were, and in future generations we were able to go back and apply those new lessons to iterative improvements and new applications.

MAN: *It is interesting that you were saying you only have four buttons and then*
BERGMAN: *everything followed logically from that. One could imagine an alternative model, and some people will argue that this is what Microsoft Windows CE does, in which you provide menus and you can have as many functions as you want. By this line of reasoning, why not just miniaturize those elements people are already familiar with in a PC world?*

ANI: Well, this is the issue of complexity. The theory driving that was what they
HAITANI teach you in first-year psychology class in college: your brain can only grasp several objects at once. If you put four dots on a page in random positions and you stare at one point, you can instantly recognize that there are four dots without having to move your eyes. But if you put nine dots in random points and stare at one, you can't tell how many there are. Maybe you can try to look at them peripherally, but that's faking it.

If you have more than the number of objects that your mind can grasp at once, your eyes go into "scanning mode." Your eyes have to travel around the page and you count the dots in a serial fashion. Now, if you have an object like a remote control with 20 buttons, in order to find the right one to press, your eye has to scan through the rows of buttons. But if you had a remote control with just four buttons, you easily memorize the location of the buttons *by feel*. There is no cognitive activity required. This makes it seem fast and easy. It's similar to when you roll dice. You don't toss the dice on the board and then count the dots—at a glance you know the total. So we felt that you simply must have a very few number of items on the screen.

Perhaps a clearer example is the hardware, where we have only four application buttons on the front, and believe me, there were a lot more apps we wanted to assign buttons to. On the original Zoomer, we had a row of about 10 icons on the bottom, and whenever you wanted to launch an app, you'd have to scan through it and find it. You'd have to concentrate. Granted

for a split second, but it still required a small amount of effort. With the PalmPilot, with only four buttons, you don't think—you just press the damn button. Again, it sounds trivial, but the cumulative effect of steps and effort adds up.

But let me get back to the concept of fewer objects. If you're trying to make a simple design on a small screen, you're not providing value to the user when you give them a whole slew of buttons. You're being lazy or indecisive, and the result of your inability to make trade-offs is a poor user experience. Let me make an analogy. When you create presentation slides, design people say you should always use big fonts because they are more legible, for example 36-point fonts rather than 8-point. Many people struggle because they want to communicate a point that in prose would take multiple sentences. If you make yourself use 36-point fonts, however, it forces you to be concise and get your point across and be very efficient in communicating. You don't do people any favors by having twice as many words on a presentation slide. The constraint forces you to be concise. It was the same with our interface design. Ten buttons are not acceptable, we must have four. It's similar to the process of editing.

I say if you only read one book to understand handheld user interface, it should be Strunk and White's *The Elements of Style*. It's a classic college reference book that liberal arts majors are more familiar with than engineers are, but it talks about how to edit extraneous words. The gist is that the fewer words you can communicate the same sentence in, the more powerful your writing is. For example, if you clean up a sentence that's in passive voice and it's 14 words, and restate it in 9 words, it's a much stronger sentence. It's very easy to be verbose, as I'm doing now just babbling on. It's easier to write a 20-page paper than it is to write a 5-page paper because it doesn't require any discipline. UI design is like an editing process—you start with a screen and one by one start ripping things out. But the key is you still have to communicate that point—you don't want to throw that out.

BERGMAN: *How do you decide what to keep and what to throw out?*

HAITANI: You have to have an understanding of what your user wants to do. What are their priorities? What's important to them? It's easy to remove buttons, but if you remove the wrong buttons, then you create a poor user experience as well. They end up searching through menus and getting frustrated. On the other hand, if you make the right decisions, then when they launch your app and they're in a hurry, they just seem to find what they need right in front of them. When it comes to advanced features, as I said people use them much less frequently, and are therefore more willing to look around for them. It's not as frustrating an experience.

A lot of people describe our design as "Zen-like." I have a tongue-in-cheek UI design presentation that uses the Zen theme. It concludes that the most important thing is to focus on the "inner tranquility of the customer." That phrase started as a joke, but the more I think of it, the more I think it's an apt analogy. Think about how you feel about PC applications. I swear at my PC all the time and I know a lot of others do as well. It gets very frustrating. You get angry and irritated when you can't do what you want to do. But if you apply the less-is-more approach, your app just does the right thing. For example, if you're in a hurry and you pull out your Palm and you want to know where you're supposed to be. You press one button and it shows you your schedule. No struggle, no effort, no confusion; it just does what you want to do. Your reaction is a contented sigh, and it's very calming. Your blood pressure doesn't rise. That's the key to the whole experience. If you can do that, then the other rules don't even matter. The rules are guidelines, but if you understand the ultimate objective and you can accomplish it by breaking all the rules, then that's great as far as I'm concerned.

:RGMAN: / **BERGMAN:** *You're describing the experience on the Palm as being in many ways fundamentally different than the PC experience. What is the philosophical difference between that environment and something like Microsoft Windows CE?*

IAITANI: / **HAITANI:** The Windows CE philosophy is that if you're familiar with the PC it takes less time to learn how to use the product. In other words, what's good for the PC is good for the handheld. Frankly, I have the opposite view because I feel you've got to look at how people use these products. At Palm we analyzed handheld usage versus laptop usage. We asked people how often they access each device, and how long they use it. We found—and this makes sense intuitively—that people access laptops relatively infrequently and for long sessions. You fire up your word processor, your spreadsheet, and you work for 20 minutes or an hour.

Handheld devices, on the other hand, are used in the opposite way. People use handhelds frequently for short bursts to quickly access data, such as to look up Joe's phone number, and then put them away. You have to ask yourself if the usage pattern is diametrically opposite, does it make sense to copy the design paradigm, and our response was no. Forget modeling a UI after something just because it's familiar. I'm familiar with how to use a door, but I wouldn't design a briefcase with a doorknob to open it. What good is familiarity if it doesn't work well?

RGMAN: / **BERGMAN:** *Was that a difficult argument to make? Did you receive much pushback from engineering or management, saying, wait a second, you want to* purposely *make this different than a PC?*

HAITANI: Not really. I never stated that we should *not* emulate a PC design. In fact, and this may sound contradictory, we tried to emulate PC design concepts as much as possible. For example, we considered more radical approaches to our menu system. It was tempting to take the opportunity to be more creative. But since we concluded our customers were PC users, we decided they would be familiar with PC concepts. But we didn't constrain ourselves to simply copying blindly for the sake of copying. We felt that too literally copying PC user interface objects was doing our customers a disservice. Instead we stripped menus down to an essence that would be recognizable, leveraging the value of familiarity. We optimized the look and behavior to be as easy and fast as possible. I think the proof is in the pudding. If the product works, and the customer achieves "inner tranquility," they don't stop to ask, "Hey, why doesn't it look exactly like my PC?" If we do our job, the question becomes irrelevant.

BERGMAN: *So given this contrast between the Palm philosophy and what Microsoft Windows CE is doing, what's your evaluation of the effectiveness of their product for its intended use?*

HAITANI: I think the evaluation of these products changes over time. When you first see one—if you're looking at a demo or playing with one in a store, you notice the familiar Windows look and feel. You may even think mistakenly that it is Windows compatible. The extra bells and whistles appeal to you. In this environment, a few delays here and there do not discourage you. As you use one over time in real-life situations, however, the sluggish performance begins to bother you.

You could write an equation that says Frequency of Operation × Time per Operation = Frustration. For example, if you boot a word processor three times a day and it takes 30 seconds, that's not a big deal. But if you use the "cut" command 50 times a day and it takes 5 seconds, that would drive you nuts. From that perspective I think people use handheld devices more like the way you use a watch than the way you use the laptop. When you check your watch, you want the time—immediately. You want to grab some information and that's it. Imagine if your watch was designed like a PC. You're late for an appointment, you pull out your watch. You press a button, and for three seconds a splash screen says, "Booting WatchOS." Next, you pull out a stylus and tap to launch the Time app. Wait cursor. Then you see a dialog that asks you to select time zone, which you thought was a neat feature when you bought it. But you almost always seem to be in the same time zone, so now it bothers you that you have to tap this every time. . . . That's obviously a silly example, but imagine one even simpler. What if you had to wait two seconds every time you checked the time on your watch. One . . . two. I guarantee over time it would drive you crazy. Try it!

In other words, we've found that people who use complex handheld devices say their utility diminishes over time. You may initially be impressed with the number of features crammed into a device. But over time, your experience drops off very quickly because (a) you're being frustrated with watching wait cursors, and (b) because you realize that you hardly use any of those features. As a result, many of these products end up in a drawer. Customer experience with Palm products has been the opposite. You pick it up, it just seems to do what you want, and you realize you don't find yourself becoming frustrated. It's just doing the job you want to do and keeping it very simple, and it's hitting that right set of functionality, and people have been very happy with it. That's why word of mouth has been so important for the Palm products.

RGMAN: *Another way where the Palm differs very strongly from the PC is that nowhere that I can think of do you see a Save button, or nowhere do you lose data because you forgot to save. How did that come about?*

AITANI: That was in from the beginning. We have no explicit concept of saving files, which is another PC concept that is not necessarily applicable to handheld devices. Think about why you want a Save button on your files. You may be editing a document, then decide that you want to scrap your changes, or go back to an earlier version. There is no such concept on, say, a date book application. You write in "Lunch with Ed" and that's it.

A related concept that we ruled out early on was the ability to have multiple files; for example, if you want to have different address books for personal, business, etc. Again we asked ourselves what a customer really wants to do. Well, I might want to have my business contacts in one file and personal contacts in another. We can solve that with categories, which would not only allow you to switch between lists more quickly, but would also allow you to display all your lists simultaneously. A good engineer could argue that a file system would be more "robust." But then you would need a file menu in each app with Save, Save As, etc. Each decision like this increases complexity and it gets worse and worse. Ironically, the end result of this minimalist thinking is that we have fast performance on a very slow processor. Conventional wisdom in the PC world dictates that the path to faster performance is to use a faster processor. It's the one-way street of constantly upgrading our computers yet always being behind. It doesn't have to be that way if the people writing the OS have some *discipline,* rather than simply adding more code because they assume that people will upgrade their machines. Jeff Hawkins simply said we should make the operating system smaller and more efficient. Another equation: Processing Speed ÷ OS Overhead = Performance. PC thinking is to simply pile on the processing

speed to address performance, but we took the opposite approach and addressed the denominator. That's just one example. We assumed we'd reject every feature unless there was a good reason not to. Saving files. Printing. By being merciless, at the end we were left with a very tight core of an operating system.

BERGMAN: *That really goes back to your earlier comment about the Japanese philosophy of hardware miniaturization being applicable to software.*

HAITANI: Yes. There's a parallel between the miniaturization and the Strunk and White approach—it all comes back to "less is more." That's because in handheld devices you have *constraints* that you don't face with a desktop PC. On the desktop, you have all the power you need because you're plugged in the wall. You have the utility company on your side, and you have a huge display, so you have plenty of screen real estate. So there's no penalty. In the PC world, 96 megs of RAM will always be better than 64 megs of RAM. You never say, well, actually I'd rather have 64 megs. You're not penalized for having an extra 50 features, so you might as well have it. Whereas on a handheld device, if you have 50 more features, in order to support them then you find your batteries last half as long, and you need a bigger screen. So you design in larger batteries, and more memory, and they start feeding off each other. And the next thing you know you have this big bloated, sluggish product. We call that the "Death Spiral."

BERGMAN: *How do you think some of the lessons you've learned in the Palm can be applied to make PCs more usable, recognizing they're different devices?*

HAITANI: That's a really good question. I don't see any reason why these lessons couldn't be applied to a PC. If you applied these same constraints to a PC, I think you could develop applications that are very easy to use. As a small example, apps with 20 icons in the tool bar drive me crazy. It's the PC mentality of more-is-better. If I can customize a tool bar, I immediately pare those back to four. Or another example is that I've always thought it would be interesting to take Windows 3.1 and Excel Version 1, load them on a 500 MHz Pentium III, and get you a blistering fast product. But the problem is on the desktop side, perhaps there is a market psychology going on, where people buy into the more-features-is-better approach. I've had people say that to me. "We tried that on the desktop; we tried a 'lite' version, and the competition kicked our ass because people didn't buy into that."

BERGMAN: *Is an alternative to that scenario to have many special-purpose devices that do particular tasks well?*

HAITANI: **IAITANI:** That's an interesting point, the concept of the "information appliance." To some degree Palm products fit into this definition because of the focused functionality we've been discussing. On the other hand, the fact is that the Palm operating system is open, and there are thousands of applications—from that perspective I wouldn't call that "focused." But the execution of any given application is very focused from the perspective of features and UI design.

But the other difference is that there's a fundamental difference between the types of applications on a handheld and a PC. One approach third-party developers take in the design of handheld applications is to take a PC application and recreate it on a handheld. That's the biggest mistake you can make! That's because, as I said, the PC and handheld are completely different animals. For example, we had an expense report application that let you track lunches and other expenses while on the road. The PC approach is to assume that you want to be able to generate reports and look at your running expense totals and calculate exchange rates on the device because that's what PC expense applications do. But that's not what you need. All you need are those features that are relevant to the mobile nature of the device because you can assume that there's a PC back at your desk. All you really want to do is jot down, say, $5 for the cab, and $12 for lunch. When you get to upload the data to your PC, you can do all the calculating there. You've got plenty of power and speed on the desktop—that's why you bought that Pentium in the first place. Let *it* do the heavy lifting.

Let me make another strained analogy. It's like designing a car. You could do some market research and find that people like to eat in the car. All right, well if we really wanted to provide a robust solution, we'd give them a refrigerator and a microwave and a range and a stove and all that. And you could. You can buy a Winnebago and drive off. But that really detracts from your driving experience. On the other hand, you can ask yourself, how much do you really eat while driving? Most of the time you drink coffee or a soda. If you really think it through and understand your customer's requirements, you decide that maybe you can just provide a cup holder. When you talk about the 80/20 rule, 80% of your customers need a cup holder, and if in order to accommodate the other 20%, you end up with a Winnebago, then forget it. Stick with your 80%, put the cup holder in, you still have your Acura. Customers ask for features like kids ask for candy. In handheld design, you have to make the trade-offs for them. Robust functionality isn't a benefit; it's a cop-out.

By the way, the analogy can be extended to describe why having a PC allows you to unburden the handheld software. You can get away with just a cupholder in the car because, by the way, you happen to live in a house with a refrigerator and a range and a microwave. That's all back in the kitchen, so

you don't have to have it everywhere you go. It's very similar to the expense app issue. You don't have to have exchange rate conversion on your expense app because you can do that much more easily on your PC.

BERGMAN: *So who out there "gets" these design concepts besides Palm?*

HAITANI: Well, obviously we do here at Handspring, since we're the ones who developed them! But seriously, I think many of the people who have tried to build handheld devices have difficulty shedding the instincts that make them successful in the desktop world. It's as if the leading supertanker manufacturers tried to design kayaks. The first thing they'd do is assemble world-class welding engineers and experts in dual-hull design. It's not a lack of intelligence—there are plenty of people much brighter than I am at our competitors. It's just a different game. At the beginning of this interview I said that I didn't have previous software experience, and I think that actually helped me. Creativity is simply an absence of assumptions, and I was lucky that I didn't have years of PC software experience to unlearn.

Obviously I'm not naive enough to assume that none of our competitors will ever figure these things out. I would argue, though, that it's easier for people outside of the PC industry who don't have the baggage of desktop preconceptions. I mentioned the Zen design concepts. I've come up with a few riddles to try to "enlighten" people on our design approach. The one people seem to remember most is a simple question: "How do you fit a mountain in a teacup?" If this question stumps you, you're still in the PC mindset.

BERGMAN: *It seems like the question you asked begs a question in return: Why would you want to put a mountain in a teacup in the first place?*

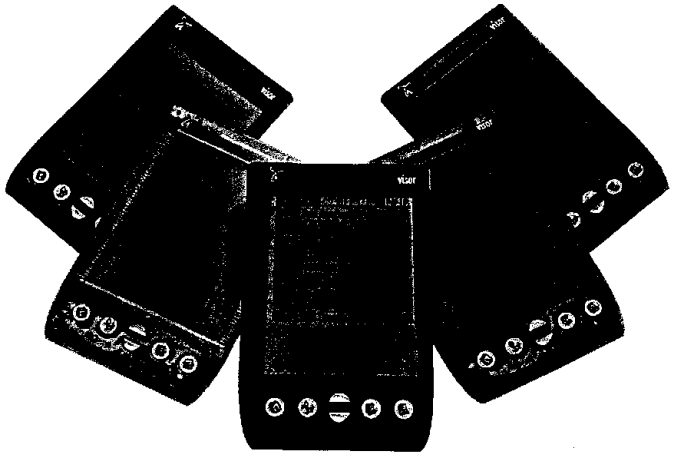
HAITANI: Exactly! If your first reaction is to try to figure out a clever way to shrink the mountain, you're taking the PC approach: *assume* we want to stuff everything in this product. You're not bothering to prioritize or distill, or ask the question about what really matters. The mountain doesn't matter.

The answer to the riddle is, Dig for the diamond and put *that* in the teacup. After all, do you really need all the dirt and rocks?

BERGMAN: *How have you applied these philosophies to your products at your new company, Handspring? Did you do something new, or are these an extension of your existing design philosophy?*

HAITANI: That's a good question. Our first product is called the Visor (see Figure 4.5), and actually we believe it takes handheld design to the next level. Basically the core of our old Palm products was simplicity—eliminating extraneous

GURE **4.5**



A spread of Visor handheld organizers.

functionality in order to create an optimal user experience. We felt that was a significant accomplishment, but in a sense it was a dead end. The Zen-of-Palm philosophy demonstrated that a simple product that worked well would succeed, and a full-featured product that worked poorly would fail. But what if you *did* want to add a certain feature? We had people all the time come to us and say that they just wanted to have a built-in pager, then they would be happy. Or a voice recorder. Or a cell phone. Individually, these are all great ideas, but if you tried to add all of them, you would end up with a big, klunky, expensive product. The key was to be able to let you customize your handheld to do what you want, when you want. So we developed the Springboard architecture, an open slot on the device that lets you plug in modules.

GMAN: *Why a new expansion architecture? What did you see as the limitations of existing technologies?*

ITANI: Well, as we learned with the PalmPilot, it's not just having a feature, it's all about execution. Backup technology existed when we invented HotSync, but we felt one-touch, no-brainer synchronization was critical if you wanted to synchronize frequently. It's the same with Springboard modules. It's not just

configurability per se. You have to make it fast, easy, flexible. Springboard modules are not "installed" or "configured." You just snap in a Springboard module and it launches the software on that module. You can insert or remove a module while applications are running. Any special drivers required to run the module are automatically installed when you insert the module and removed when you pull it out. This eliminates the problem of conflicting drivers that crash your machine. No resets, no crashes, nothing "bad" happens. This lets us keep the base OS lean and mean, but provides the flexibility to add whatever specific functionality you want. That's what we see as the next big step in advancing handheld computer design.