Designing a New Form Factor for Wearable Computing

Despite significant improvements in underlying technologies, the wearable computing field is still in its youth. This article offers several methods that can help accelerate the process from vision to product.

Chandra Narayanaswami and M.T. Raghunath
IBM TJ Watson Research Center

Wearable computers have more to do with form factor and usability than with computing.[1,2] Aside from a few early adopters who might be eager to experiment, most potential users are primarily interested in what a wearable device can do for them. These potential users must be convinced that the benefits of wearing the device outweigh the difficulty involved in carrying or caring for it.

You can measure a wearable device's usefulness by its form factor, usability, and applications. A device's form factor defines its size, its shape, and how users wear or carry it. Usability entails not only how intuitive and easy the device is to use but also how it is used—with one hand, for example, or through an audio–visual interface. Ultimately, a device's applications will determine whether consumers will buy and use it. A device's applications might let users perform some tasks more easily than without the device. Or they might let users perform tasks not possible without the device.

On the other hand, the primary factors hindering the adoption of wearable devices include weight, size, price, and battery life. As technology improves the ability to miniaturize circuits, designers can reduce the weight, size, and price of wearable devices. And efficient energy use[3] and better energy technologies can help prolong battery life.[4,5]

However, despite significant improvements in underlying technologies, the wearable computing field is still in its youth. While researchers constantly come up with new visions and ideas,[6,7] the path from vision to product is often difficult to navigate. For example, it is hard to justify the investment in building a product unless you can be reasonably sure that the product will make money. This problem is even more acute in the current investment climate, in which funding for experimental projects is extremely limited. Even after a company develops a marketable product, many products fail to get past the early-adopter phase.[8]

This article—based on our experience in building the Linux Wristwatch at IBM[9]—offers several ideas that can help accelerate the process from vision to product.

Rapid prototyping

Although many of the ideas presented here are tailored to wearable form factors, these ideas are similar to other approaches for rapid prototyping. For instance, rapid mechanical design prototyping is a mature methodology. Several technologies, such as stereolithography, were created specifically to aid in rapid prototyping. Paul Kenneth Wright examines the uses of rapid design, development, and manufacture of complete systems in a recent book.[10] Other efforts address rapid prototyping of integrated circuits, physical models, mechanical assemblies, and other technologies.[11]

Developers have used similar ideas in object-oriented software[12] to speed the process of developing

1536-1268/02/$17.00 © 2002 IEEE

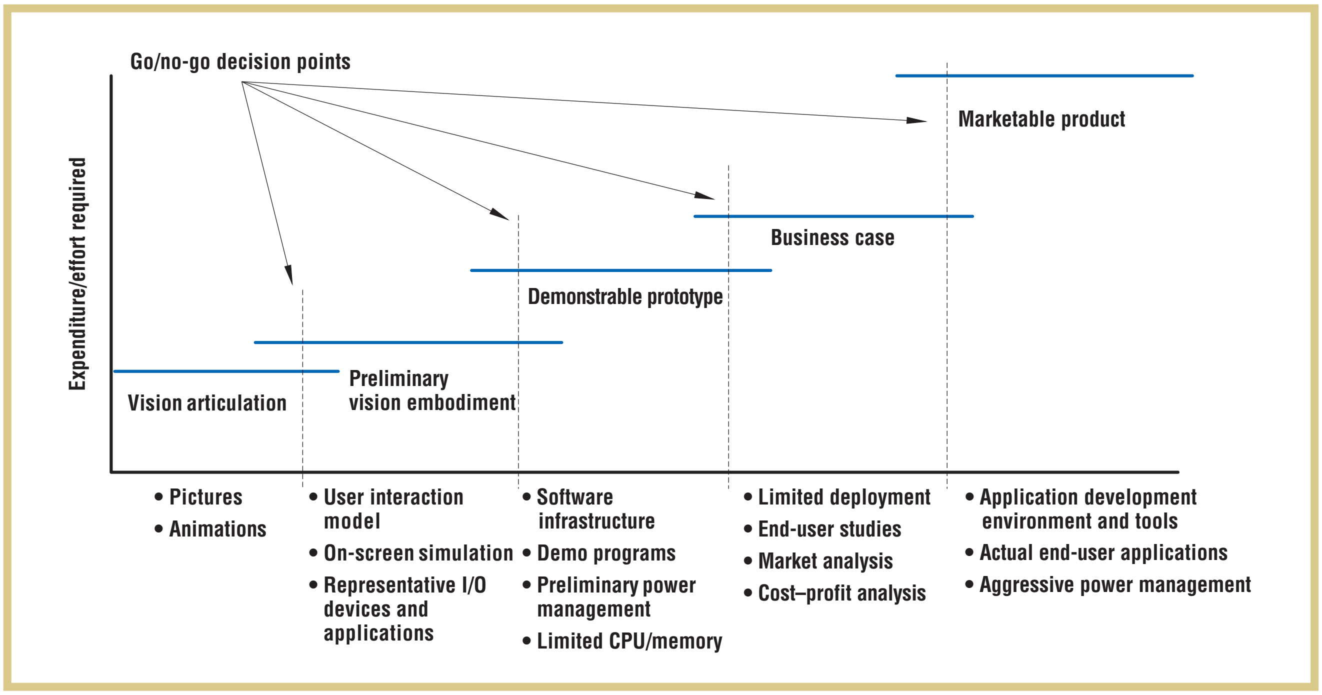


Figure 1. The steps from vision to product include vision articulation, vision embodiment, demonstrable prototype, business case, and marketable product.

complex software systems with design patterns. Processor designers have used software simulators to see how increasing the number of arithmetic units, adding new instructions, and changing cache sizes affects performance. Other researchers have shared their pioneering experience in building and deploying several generations of wearable computers.[13]

Clearly, we can use existing rapid-prototyping approaches to speed the individual steps of our process, but our approach augments prior work by examining the overall system design. It starts with a tailored user-interaction client and a general-purpose server that executes the functions, then progresses to a complete prototype and shows the interactions between the phases. Although we emphasize wearable devices, lessons from this research apply to other mobile-computing devices, which present similar challenges.[14]

Steps along the path

Figure 1 shows the steps involved in building a new wearable device. We organize the steps into five phases (similar to those Asim Smailagic and Daniel P. Siewiorek describe[15]). Although presented sequentially, we often iterate through each phase. We might have to revisit decisions from earlier phases on the basis of what we learn from later phases.

Vision articulation

A new device typically begins as a vision, which might come from thinking about the problem in isolation or from interacting with users who provide the inventor with insights about user needs. The inventors have to develop the idea until they are convinced it is workable. This phase's objective is to motivate a small core team to take the idea to the next phase.

The team must be able to provide answers to several basic questions: What is the product and what will it do? Is it feasible to build it? Who are the potential users? How will they use it? How much will it cost? How will it look? At this early stage, it might be hard to have clear answers to all these questions, and the answers might change as the project progresses. However, it is important to collect and validate the answers with a target audience.[16]

Getting support for the vision is crucial before moving to the next phase. It is also important at this stage to study several related, commercially available devices to learn from other people's experience and to be able to present the advances in vision clearly.

Preliminary vision embodiment

The next phase in the journey is to build a preliminary embodiment of the vision that will be close to the final look and feel presented to the end user. This helps clarify the vision to make important design decisions. Developers can build multiple embodiments of the prototype that consist of plastic or paper mock-ups, or even something that can actually model user interaction.

Embodiments should be as close as possible to the final device in terms of its user-visible elements, such as display screens, buttons, and dimensions. It is not critical to build the actual device or the complete applications that will run on the device because doing so is more expensive and

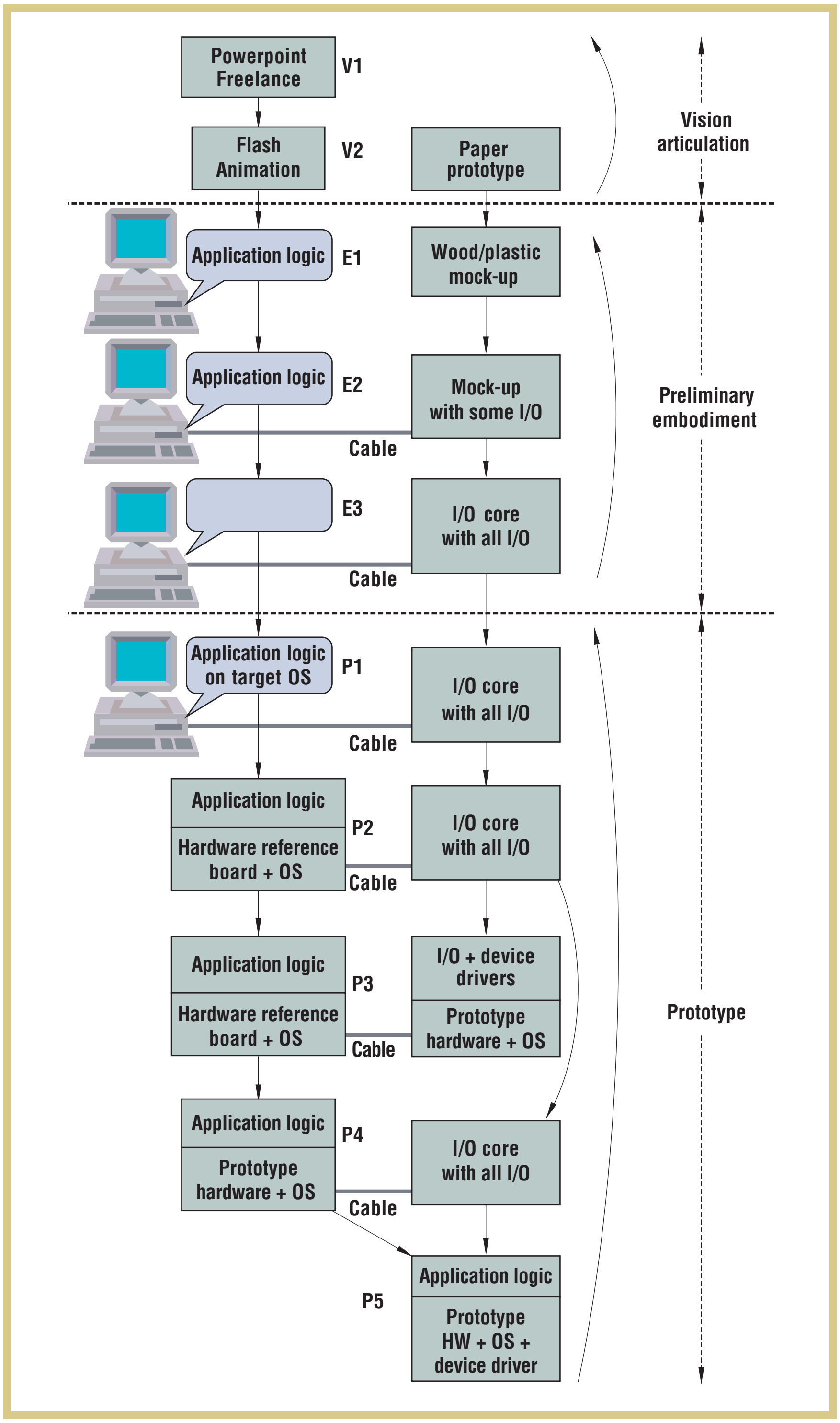


Figure 2. This detailed breakdown of the path to the prototype shows the iterations within each step of the process.

will typically happen much later in the process.

Demonstrable prototype

If the preliminary embodiment is convincing and there is agreement to take the process further, the next phase entails building a working prototype. At this stage, a design team validates the assumption that the product is feasible. Another objective is to create demonstrations that show the device's capabilities. The demonstrations should allow an outsider, who has no prior knowledge of the device, to wear it and quickly understand its purpose and how to use it.

The prototype helps refine several key usability aspects of the device. Although the design team can try to visualize how a device will be used and can try to verify it through the preliminary embodiments, the prototype provides real validation of device usability. As the project progresses through these stages, new ideas will continue to emerge. It is important to incorporate some level of flexibility in the prototype and to address the usability issues that might emerge during demonstrations.

Business case and marketable product

Out of many ideas that make it to the prototype phase, some of them do evolve into a marketable product. More effort is required to go from a prototype to a product than from an idea to a prototype—and doing so requires a different skill set from the first three phases.

It is often difficult to analyze a business case without a working prototype. Building a business case is also easier after the design team deploys a limited number of prototypes among a friendly audience to assess the responses. This exercise, which is similar to software beta testing, can help find some of the prototype's shortcomings that the team overlooked in the design process. Phases four and five are outside the scope of this article but are addressed by several books.[17,18]

Accelerating the path to prototype

Figure 2 shows our suggested approach to breaking down the first three phases. We designed our approach to ensure the maximum use of existing tools, including using general-purpose computer hardware and software to reduce time and cost. Several tools and methods can help build storyboards and physical mockups to speed up the early phases. The rapid growth of processing power in general-purpose personal computers and the availability of wireless connectivity can help speed the preliminary embodiment and prototyping phases.

Design teams can study and improve several important features of the target device by first building simpler user-interaction cores that embody the target device's key I/O characteristics. These interaction cores connect to general-purpose computers that actually run the software intelligence. Our approach uses a client-server model to speed the prototyping process. The interaction core helps evaluate the usability and interaction issues. Using lessons from these activities, a design team can build the first functional prototype while software development continues on the user interaction cores and general-purpose computers or other development systems.

We describe the three phases sequentially, with iterations between the phases to simplify the presentation. In practice, however, the design cycles are more complex and iterative. The exact nature of design iterations will depend on which generation of product is being built, how the market conditions change along the phases, and so on. To facilitate understanding, we divide the three phases into smaller steps, labeled V1–V2, E1–E3, and P1–P5.

Vision articulation

Graphical animations and paper prototypes (step V2) should show how the device looks in 3D. In conjunction with storyboards,[19] these mockups should also articulate how users will wear the device, in what contexts it will be used, and by what kind of users. Fortunately, many software tools make it possible to create detailed and visually appealing presentations without a great deal of effort. Inexpensive digital cameras and camcorders can also help the designers with storyboards and can help present the user's environment.

In addition, designers can simulate user environments.[20] For example, if a device is going to be used by a ticket conductor on a train, it would be useful to know how crowded the trains are, whether people are sitting or standing, how long the average ride is, and what the lighting conditions are. The development team can make several design decisions even at this early stage. For instance, the team can select fonts to improve the readability of information presented on the device and can estimate the wireless bandwidth requirements.

Multiple iterations at this stage are easy, cost effective, and useful in refining the vision and explaining that vision to others. The team must resist the temptation to move on to the subsequent phases too quickly.

Accelerating the embodiment

The first and simplest embodiment to build is a plastic or wood mockup (E1) of the actual device. Having something that you can touch will help refine many aspects of the device, such as the shape and positioning of its controls.

Along with the physical mock-up, it is also useful to model the actual user interaction with an on-screen visual simulation model of the device on a desktop computer. The computer presents the device's actual user interface elements[21] on the screen, while the computer's keyboard and mouse simulate the device's UI elements. If the target device will have I/O devices—such as an accelerometer or a GPS receiver—that are hard to simulate using the keyboard and mouse, you can connect these I/O devices directly to the desktop computer.

The visual simulation model should react in the manner the actual device is expected to react, displaying the appropriate screens. To keep the on-screen model flexible enough to add new ideas, the team should build the application logic for the on-screen model using high-level programming tools. The team should also keep the on-screen model simple and should avoid modeling the internals of the target device.

You can create simplistic on-screen models by tweaking existing emulators for other devices. For instance, if an emulator for a handheld device or a cell phone is available, you could consider replacing its background image with that of the target device. If you cannot modify an existing emulator to emulate the target device's main aspects, you will have to build a custom emulator. We initially considered modifying a cellphone emulator, capable of displaying WML (Wireless Markup Language) content, to build the first on-screen model for our Linux Wristwatch. However, because such emulators could not generate graphics required to display a clock face, we developed a more custom on-screen prototype using Tcl and Tk scripts.

The next step (E2) is to bring these two separate embodiments together into one embodiment that is comparable to the actual device in dimensions and also reacts to user input the way the actual device would. The best way to accomplish this difficult step is to think of the embodiment as a thin client containing just the I/O devices, with the desktop computer as the server that runs the application logic. Alternatively, you can use a small but powerful portable system such as the IBM MetaPad[22] as the server.

In step E2, we add a few of the crucial I/O devices to the mock-up and connect these I/O devices to the application logic that continues to run on the desktop computer. In many cases, you can use standard connectors to interface the I/O devices to the desktop computer. For example, we put a small VGA display into our wristwatch mock-up. A module that accepted a standard monitor cable served as the driver. Likewise, you can connect input buttons to the parallel port. In some cases, the I/O device might connect to the embodiment using a different interface than the one the prototype uses. For example, the fingerprint sensor in our watch was also available as a USB attachment. We used the USB attachment for the embodiment and a direct connection for the prototype.

At this time, the design team should create a software and hardware specification so that the detailed hardware design can get started and the team can initiate contact with hardware vendors to get quotes on cost and schedule. It is important to understand when making these decisions that limited volumes could mean a slower schedule. To the extent that it is possible,

TABLE 1
Matching embodiment to prototype.

Constrained feature	How to handicap the embodiment
Display pixel pitch	Use pixel blocking
Display brightness	View under bright ambient light
CPU speed	Add delay loops or run on a slower processor
Floating point	Avoid using floating point
Memory size	Monitor or limit the memory usage
Memory latency	Modify the compiler output, adding a delay loop before each load
Network bandwidth	Add some overhead to the network stack
Touch screen resolution	Use blocking
Speaker	Feed some noise into the speaker
Microphone	Add noise to the prototype microphone.
Weight	Add some weight to the I/O core

it is preferable at this stage to select the final I/O components and sensors and use these components in the core. Doing so can help establish a lower bound on the prototype device's size and can also help with its power estimates.

As the team adds more I/O devices to the mockup (step E3), it is advantageous to add a small I/O controller to manage these devices. As a result, the mockup becomes an I/O core, and the team must define a more complex cable connection and a protocol for data exchange over the connection. Designing this protocol can be challenging. Although the protocol must be simple enough to execute on the small I/O controller managing the I/O devices in the core, it must be able to handle bidirectional information flow and also multiplexing data from multiple I/O devices. Depending on the level of complexity, it might be necessary to subdivide the I/O devices into multiple classes, with each class having its own I/O control logic.

Instead of building custom circuitry in this step, it might be possible to leverage other small form-factor prototypes that the design team might have built in the past and then replace the I/O devices with the ones for the target device. For instance, you could take apart a prior generation handheld prototype and replace its I/O devices to create the I/O core. The handheld processor could then control the new I/O devices. The interface connector in this case might be as simple as a serial or USB connector.

At the end of the embodiment phase, the team should have a thin-client embodiment that does most of what the target device will do but is driven by application logic running on a standard desktop computer. To make sure the embodiment sets realistic expectations, it is a good idea to model some of the resource constraints that are likely to be present in the prototype or the final product. In general, if the desktop computer or the I/O core is more capable than the target device, these capabilities should be handicapped in the embodiment.

Table 1 discusses how we might match the capabilities of the I/O core to that of the target device. For instance, if the display in the embodiment has a much finer pixel pitch, you can group the pixels so that a square area corresponding to four or nine pixels in the embodiment will emulate one pixel in the target device.

Although unlikely, the target device might have better capabilities than the embodiment. In this case, the design team must either improve the embodiment's capabilities or appeal to the user's imagination to fill in the gaps. For example, you can emulate a bright display on the target device by viewing the display in a dark room.

Accelerating the prototype

Assuming the results from the embodiment are encouraging, the team must make several fundamental decisions before building the prototype. What processor will power the prototype, how much memory will it have, and what operating system will it run? In our project, we deferred these decisions to this stage so we could make these choices according to a just-in-time development model, which is important because these decisions do not affect the user's perception of the device's functionality. Also, deferring these decisions until later lets us take advantage of newer hardware components that are likely to be less expensive and more efficient.

The experience and tests with the embodiment can provide valuable input into the amount of memory required, how much processing power is needed, and so forth. Once the team makes these decisions, it can then begin the hardware design for the prototype. As the hardware design progresses, the application logic should be modified so that it can run on the target operating system (step P1) with the target operating system running on a desktop computer. The modified application logic should preferably use the same source-level APIs on the desktop computer as it will on the prototype. The desktop computer's CPU architecture might differ from that of the prototype, but that is not of concern at this stage. Some embedded real-time operating systems might not be easy to simulate on a desktop computer. If so, you might want to skip ahead to step P2.

Vendors that design processors suitable for wearable computers usually provide reference or development boards. They might also provide an operating system to run on such boards. The next step (P2) is to bring up the desired operating system on the development board if the board vendor has not already done it. The team follows this step by migrating the application logic to the development board. The development board from the processor vendor should be more stable than the prototype hardware that we build, and it typically has other interfaces and tools that can aid in bringing up the system. If the development board is not capable of interfacing to the I/O core directly, the team might have to build special interface hardware and software.

Once the prototype hardware is available, the first step is to bring up the OS and drivers that handle the I/O devices on the prototype. As shown in step P3, a good way to test the I/O devices is to treat the prototype hardware as if it were the I/O core used in step P2 and interface with the same server-side code. The server still runs all the application logic, but the actual prototype handles the interaction. Depending on the kind of communication interfaces supported by the prototype, any one of the server-side pieces from steps E2 to P2 could drive it.

Another way of checking the prototype hardware is to use it as a replacement for the development board. You would run the application logic on it and present the user interface on the I/O core (step P4). However, this step requires the ability to connect the server side of the cable connection to the prototype, but you can also do this without developing all the device drivers on the prototype. Depending on which is easier, you might need only one of the steps, P3 or P4.

The final step in the prototype is to have all the I/O code and the application logic execute on the prototype hardware. At step P3, the hardware development board might have more resources than the prototype, which means that moving from P3 to P4 or P5 might require that the design team address resource constraints. If the team addressed resource constraints earlier in the process, there should be fewer surprises at this stage. Depending on the device being prototyped, it might be possible to replace the cable in steps P1 to P4 with a wireless connection. For example, if a team were prototyping a new glove for user input, using a wireless link might be feasible because the glove would likely only require a small amount bandwidth.

If some of the prototype's features do not work because of hardware bugs, the team can connect prototype P5 to P1 and simulate the defective feature in software. Alternatively, the team can simulate features that it dropped from this version of the prototype. Teams can also use this method to determine how much more processing power must be added to the prototype for any new function.

Although we described the steps sequentially, there is considerable overlap between the development activities associated with the different steps. For instance, the hardware design for the prototype hardware of step P3 will typically begin early, perhaps around step E2 or E3, given the turnaround time for building the hardware. It is important to leverage existing rapid-prototyping methods to reduce the hardware design cycle.

The amount of time spent in each stage depends on several factors. For example, with the first version of the Linux Watch prototype, the total time from the vision stage to the working prototype stage was about two years. We spent roughly 5 percent of the total time on vision articulation, 40 percent on preliminary embodiment, 30 percent on prototype hardware building and testing, and 25 percent on moving the software from the embodiment to the prototype. We spent about 5 percent of the total effort in the vision articulation stage, 30 percent on the preliminary embodiment hardware and software, 40 percent on building and testing the hardware prototype, and 25 percent on the software (including applications for demonstration). For the most recent version of the Linux Wristwatch prototype, we added a fingerprint sensor, a two-axis accelerometer, a battery that has five times the capacity, a battery gauge, three small dedicated buttons, and twice the memory capacity. Refinements in software included power management, networking, mobility, and security.

Beyond the first prototype

As designers move to the first prototype, they gain clarity on several aspects of the targeted wearable device. Typically, they expose the prototype to several people in the target audience to get a better idea about whether the proposed form factor would be suitable to deliver the applications to the end user. The team would also be in a position to assess the target device's usability, comfort, power consumption, and functionality.

At this point, the design team faces a decision as to what the next steps ought to be. One possibility is that the prototype might have been well received, which could lead to deciding to build a product based on the prototype. There are different trade-offs in building a prototype versus building a product. The cost of each unit is a critical aspect of the product. The design team might also need to build robust versions of the applications for the product device. Application developers might be able to use the prototype devices to develop and debug the applications. And at this point, the team might also use instruction-level and functional simulators in place of actual prototype hardware to enable a larger team of developers. Using such a strategy might help reduce the debugging effort, but this approach won't work too well if the simulator is too slow. The team will need to expend additional effort to optimize various aspects of the device, such as power management, code efficiency, and code size.

Another potential option at this stage entails the team deciding that the basic idea is not really workable and will not result in a product that can be taken to market. Hopefully, the design team will arrive at this decision earlier in the development process.

Other decisions include determining whether to go through another iteration and build an improved prototype. In this case, instead of starting afresh, the team might be able to use the first prototype as the initial I/O core for the second prototype. If the team engineered the first prototype with flexibility in mind, it might be able to transform the first prototype into an I/O core for the second prototype by replacing the requisite I/O.

Reusing the first prototype to build the second prototype's I/O core might not work if the second prototype's form factor will be substantially smaller than the first one. If the decision to build a second prototype is largely motivated by the need to build something smaller, the team should keep in mind that any prototype it builds will likely be larger and less elegant than the product version, largely because the product comes later and the available technology will likely make further miniaturization possible.

The design team can limit the impact of the size difference between the prototype and the final product by using more expensive and advanced packaging technologies for the prototype.

The stepwise-refinement approach we've discussed can help innovators make informed design decisions on the basis of actual user experiences obtained early in the design process. Design teams can defer decisions that do not affect user perception until later stages in the process. This staged approach has the potential to reduce the required resources and accelerate the prototyping process.

We also discussed ways in which the product development team can leverage the intermediate systems to get a head start on efforts to develop a marketable product. We hope innovators will use our approach to convert more of their visions to working prototypes and put them in the hands of business development teams, resulting in better-designed products. ◼

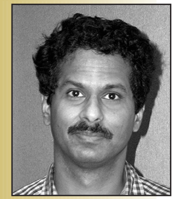
ACKNOWLEDGMENTS

We thank all the members of the Linux Watch team at IBM for their work on creating two generations of wristwatch computer prototypes. We also thank the anonymous reviewers for their comments and suggestions on earlier versions of this article.

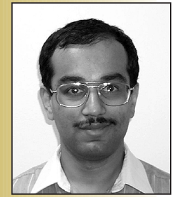
REFERENCES

1. T. Starner, "The Challenges of Wearable Computing: Parts 1 and 2," *IEEE Micro*, vol. 21, no. 4, July 2001, pp. 44–52 and 54–67.
2. D.P. Siewiorek, "Wearable Computing Comes of Age," *Computer*, vol. 32, no. 5, 1999, pp. 82–83.
3. M. Weiser et al., "Scheduling for Reduced CPU Energy," *Usenix Symp. on Operating Systems Design and Implementation*, Nov. 1994, pp. 13–23.
4. T. Starner, "Human Powered Wearable Computing," *IBM Systems J.*, vol. 35, no. 3, 1996, pp. 618–629.
5. N. Kamijoh et al., "Energy Trade-Offs in the IBM Wristwatch Computer," *Proc. 5th IEEE Int'l Symp. Wearable Computers*, IEEE CS Press, Los Alamitos, Calif., 2001, pp 133–140.
6. R.Want et al., "An Overview of the Parctab Ubiquitous Computing Experiment," *IEEE Personal Communications*, vol. 2., no. 6, Dec. 1995, pp 28–43.
7. W.R. Hamburgen et al., "Itsy: Stretching the Bounds of Mobile Computing," *Computer*, vol. 34, no. 4, 2001, pp. 28–36.
8. G. Moore and R. McKenna, *Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers*, Harper Business, New York, 1999.
9. C. Narayanaswami et al., "IBM's Linux Watch: The Challenge of Miniaturization," *Computer*, vol. 33, no. 1, Jan. 2002, pp 33–41.
10. P.K. Wright, *21st Century Manufacturing*, Prentice Hall, Upper Saddle River, N.J., 2001.
11. "Shortening the Path from Specification to Prototype," *Proc. 7th IEEE Int'l Workshop on Rapid System Prototyping*, IEEE CS Press, Los Alamitos, Calif., 1996.
12. E. Gamma, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, 1995.
13. S. Finger et al., "Rapid Design and Manufacture of Wearable Computers," *Comm. ACM*, vol. 39, no. 2, Feb. 1996, pp. 63–70.
14. M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Personal Comm.*, vol. 8, no. 4, Aug. 2001, pp 10–17.
15. A. Smailagic and D.P. Siewiorek, "System Level Design as Applied to CMU Wearable Computers," *J. VLSI Signal Processing Systems*, vol. 21, no. 3, 1999, pp. 251–263.
16. A. Smailagic and D. Siewiorek, "User-Centered Interdisciplinary Design of Wearable Computers," *ACM Mobile Computing and Comm. Rev.*, vol. 3, no. 3, 1999, pp 43–52.
17. B. Bysinger and K. Knight, *Investing in Information Technology: A Decision-Making Guide for Business and Technology Managers*, John Wiley & Sons, New York, 1996.
18. D. Remenyi, *IT Investment: Making a Business Case*, Digital Press, 1999.
19. S. J. Andriole, *Rapid Application Prototyping: The Storyboard Approach to User Requirements Analysis*, John Wiley & Sons, New York, 1993.
20. John Barton et al., "Ubiwise Simulator for Ubiquitous Computing," 2002; www.hpl.hp.com/personal/John_Barton/ur/ubiwise/index.htm.
21. J. Wilson and D. Rosenberg, "Rapid Prototyping for User Interface Design," *Handbook of Human-Computer Interaction*, M. Helander, ed., North-Holland, New York, 1993, pp. 859–875.
22. IBM, "IBM Research Demonstrates 9-Ounce Prototype Portable Computer to Explore Future Devices," *IBM Research News*, 2002, www.research.ibm.com/resources/news/20020206_metapad.shtml.

the AUTHORS



Chandra Narayanaswami is a manager leading a wearable computing group at the IBM T.J. Watson Research Center. His current interests include wearable computing, energy management, applications, and user interfaces for pervasive devices. He received a PhD in computer and systems engineering from Rensselaer Polytechnic Institute. You can reach Narayanaswami at IBM T.J. Watson Research Center, 19 Skyline Dr., Hawthorne, NY 10532; chandras@us.ibm.com.



M.T. Raghunath is a research staff member at the IBM T.J. Watson Research Center. His current interests include wearable computing, embedded operating systems, applications, middleware, and user interfaces for small form-factor devices. He received a PhD in computer science from the University of California, Berkeley. You can reach Raghunath at IBM T.J. Watson Research Center, 19 Skyline Dr., Hawthorne, NY 10532; mtr@us.ibm.com.