Designing for the Digital Age: How to Create Human-Centered Products and Services, Kim Goodwin

Designing Interfaces: Patterns for Effective Interaction Design, Jenifer Tidwell

Designing Gestural Interfaces, Dan Saffer

Designing Web Interfaces: Principles and Patterns for Rich Interactions, Bill Scott and Theresa Neil

Mobile Interaction Design, Matt Jones and Gary Marsden

Communicating Design: Developing Web Site Documentation for Design and Planning, Dan Brown

Designing the Obvious: A Common Sense Approach to Web Application Design, Robert Hoekman Jr.

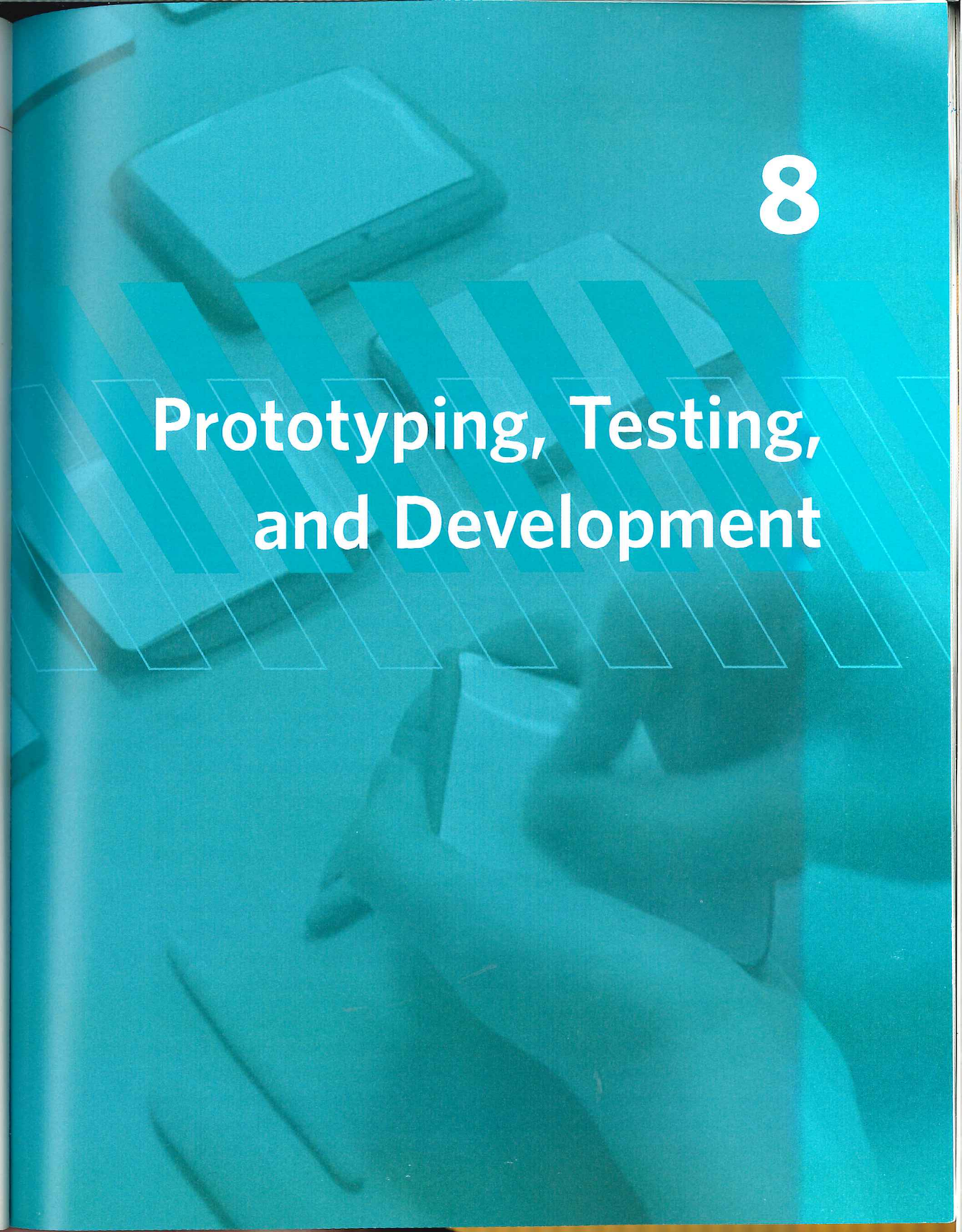
Information Architecture for the World Wide Web: Designing Large-Scale Web Sites, Louis Rosenfeld and Peter Morville

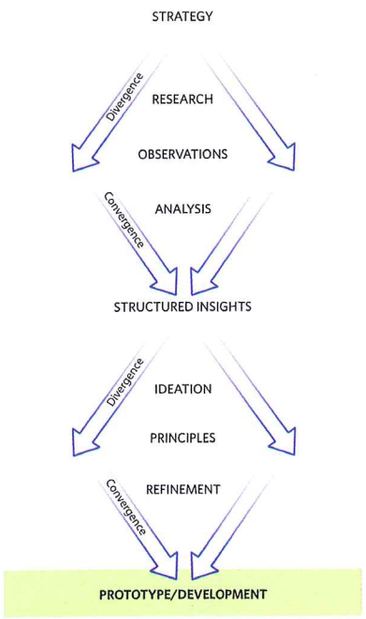
Ambient Findability: What We Find Changes Who We Become, Peter Morville

Content Strategy for the Web, Kristina Halvorson

8

Prototyping, Testing, and Development





Once you have worked out some or many of the details (or even before), it is time to prototype and actually build the product. This phase of design is about refining the parts of the design that cannot be easily done with paper documentation, namely timing, animation, movement, and interaction.

One of the first things you must consider when prototyping is the interface design.

Interface Design

We can engage with digital products only through some sort of interface. Although there have been some strides in the field of Brain-Computer Interface, most of us cannot yet connect to digital devices through a cable directly from our brains to microprocessors. For now, we need some intermediary to communicate between us and our digital devices: an interface.

Interface design is so closely tied to interaction design that many believe they are the same thing, which isn't exactly true. Interface design is the *experienced representation* of the interaction design, not the interaction design itself. The interface is what people see, hear, or feel, and while it is immensely important, it is only a part of interaction design.

Digital products are a bit like icebergs. The part that can be seen (the interface) is really just the tip; what's below the surface, what isn't seen, is where the main part of the interaction design lies: the design decisions that the designer has made and the technical underpinnings that make the interface a reality. An interface is where the interaction designer's choices about how people can engage with a product and how that product or service should respond are realized. In other words, the interface is where the invisible functionality of a product is made visible (often via affordances—see Chapter 7 for more information), accessible, and usable.

In the past, form often closely followed function. A hammer looks the way it does because its shape is optimal for driving in nails. With digital devices, however, form doesn't necessarily follow function. Objects on a screen can have any shape and can potentially serve any purpose. For example, an unlabeled button sitting in the middle of a Web page could look like an

elephant, a tea cup, or even like, well, a button, and clicking it could open another Web page, start an animation, play music, close the browser window, or do a variety of other things. Likewise, the physical form of an object may have nothing to do with the behavior it can exhibit. A round orb can do anything from control your TV to alert you that a stock is falling. When working with digital devices, the interaction designer has a lot more fluidity and ambiguity to account for.

What interaction designers are most concerned about with interface design is generally the layout and placement of controls and navigation. For hardware/software products, a functional cartography (see Chapter 7) should be performed to figure out what controls go where.

Onscreen, designers need to provide cues regarding where the user should look. Color can be used to attract the eye, as can contrasting fonts (larger, bold, and so on). Lines and boxes can group objects together, but these should be used sparingly, lest the users focus on the lines and boxes and not the features—try to use whitespace instead.

In the Western world, the eye generally travels from left to right, top to bottom, and designers should be aware of this flow and design for it. Don't force users' eyes to jump all over the screen.

When objects are close together, Gestalt psychology tells us, the mind will assume that they are related. This is a good thing when designers want objects to seem related—for example, a Submit button next to a text box—but not so good when the pieces of functionality drifting into each other are unrelated.

Positioning and alignment of objects are also important. Objects that are aligned will appear to be related, and objects should ideally be aligned horizontally and vertically to create a clean look. Objects that are indented beneath other objects will appear to be subordinate to those above them, and objects near the top of the screen will generally seem more important than those farther down.

Designers should always perform the squint test on their visual interfaces. By squinting at the screen, designers can optically smudge the details and see which items on the screen have prominence. This test can often lead to surprise, revealing that secondary or unimportant items seem overly important in the design. The squint test helps ensure that the layout is strong.

Luke Wroblewski on Interface Design

Luke Wroblewski is an interface designer, strategist, and author of the books Site-Seeing: A Visual Approach to Web Usability *and* Web Form Design. *He is currently Senior Director of Product Ideation & Design at Yahoo!.*

How can visual design support (or detract) from interaction design?

In most applications, audio cues need to be used sparingly and instructional text is rarely read. As a result, the visual design bears the responsibility of communicating the possibilities, limitations, and state of interactions. It tells users what they are seeing, how it works, and why they should care.

When visual elements are applied without an understanding of the underlying interactions they are meant to support, however, the wrong message may be sent to users. Visual styling that obscures or clouds crucial interaction options, barriers, or status messages can have a significantly negative impact on user experience.

Think of visual design as the "voice" of interaction design and information architecture. It communicates the importance of (and relationships between) the content and actions within an application.

What do interaction designers need to know about visual design?

Visual design can be thought of as two interwoven parts: visual organization and personality. Visual organization utilizes the principles of perception (how we make sense of what we see) to construct a visual narrative. Through applications of contrast, visual designers can communicate the steps required to complete a task, the relationships between information, or the hierarchy between interface elements. So clearly visual organization is a key component for successful interface designs.

Unfortunately, most discussions about the effectiveness of visual design don't focus on visual organization systems. Instead, they are limited to a subjective analysis of the personality (look and feel) of an interface. Personality is achieved through a judicious selection of

Luke Wroblewski on Interface Design *(continued)*

colors, fonts, patterns, images, and visual elements designed to communicate a particular message to an audience. But just about everyone has a color or font preference, so when asked to evaluate visual design that's where they turn first.

My advice to interaction designers is to take the time to learn the principles underlying visual organization. You'll be better able to communicate with the visual designers on your team and, more importantly, with the end users of your product.

What are some of the common interface mistakes that new interaction designers make?

The most common interface design mistakes I see are overstatements of visual contrast. For example, designers will want to make sure everything on a screen can be found and therefore apply an equal amount of visual weight to each element to ensure it's "discoverable." The problem is when every element on a screen is shouting to get noticed, no one gets heard. As a user, you can recognize these types of designs because your eyes bounce from one element to the next. There is no hierarchy and as a result no flow through the content and actions on the screen.

Similarly, many designers will overemphasize the differences between individual interface elements through multiple visual relationships: different font, size, color, and alignment. You don't need excess visual contrast to distinguish objects or make things findable. Think about ways to "eliminate the unnecessary so that the necessary may speak" and aim for the least effective difference between elements.

You talk a lot about personality. How do you provide a visual personality to your designs?

Whether you've thought about it or not, people will ascribe a personality to your product based on the way it looks and acts. Therefore, it is in your best interest to be aware of the personality you are creating for your site through visual design (or lack of it) and make certain it is telling the story you want.

Luckily, there's a huge visual vocabulary available for establishing an appropriate personality for your application. Millions of colors, hundreds of thousands of font choices, and innumerable patterns and images are all at your disposal. The trick is settling on the right combination of these for your particular needs. Consider what you want to communicate to your audience and how; then locate visual elements that convey that message in the world around you. You'll be surprised at what you can find when you look!

Sound Effects

Sound is both over- and under-used in interaction design. Nearly everyone has had the experience of going to a Web site only to have it suddenly blast music, sending you scrambling to turn the thing off. But sound effects, done well, can subtly enhance an interface.

Sounds can be ambient cues that something has happened so that users don't have to constantly monitor the application for changes. This use of sound is especially helpful in applications with frequent changes that may occur while the user is otherwise occupied. A ding indicates that an e-mail has arrived. The door-opening sound indicates that a buddy has signed onto the instant messenger client. The ring of a mobile phone indicates that a text message has arrived. These are all helpful sound cues.

How can you tell if a sound will, over time, become an annoyance? Record it. Test it on others and see what they think. Listen to it frequently. Use the application yourself and see if you become annoyed at it. If you do, probably others will as well.

Prototyping

Aside from the finished product, prototypes are the ultimate expression of the interaction designer's vision. The importance of prototypes cannot be overestimated. Indeed, many designers feel that prototyping is *the* design activity, that everything before it is but a prelude, and that to design *is* to prototype.

Prototyping is where, finally, all the pieces of the design come together in a holistic unit. Indeed, many people will have difficulty understanding a design until they see and use the prototype. Like all the other models and diagrams, they are a tool for communicating. Prototypes communicate the message "This is what it could be like."

What form these prototypes take depends on both the designer's resources and the type of product or service that is being designed. A designer with the right resources can produce some high-fidelity prototypes that look and behave just like the final product or service would. Many retail chains build whole prototype stores, for example.

Todd Zaki Warfel on Prototyping

Todd Zaki Warfel is a founding partner at Messagefirst, where he focuses on design and research for consumer and Business-to-Business products. With over 16 years of industry experience, Todd has been fortunate enough to create over 15 industry-first products. He is the author of the book A Practitioner's Guide to Prototyping.

Why should interaction designers prototype anything? Why not just jump from concept to development?

Prototyping is a great way to work through your design concepts. With wireframes and Photoshop comps, the interactions are missing. This is especially problematic with dynamic transitions. Instead of being able to show the actual interaction, you're left to describe. In lieu of a prototype, I've often found myself whiteboarding and waving my hands in the air to describe a particular transition. I find it easier just to pick up some paper, draw on it, fold and tear it, and make a mini prototype to show the intended interaction.

When jumping directly from concept to development, you risk leaving the interpretation of your intended interaction up to engineers. They are more likely to take the route of least resistance (a.k.a easiest to code), which is most likely not the originally intended interaction.

The biggest benefit—besides working through your design—to prototyping? You create a clearer vision among the team. You get to show and tell and not rely on imagination and misinterpretation.

What should interaction designers prototype?

You don't have to prototype everything—it's a prototype. And prototypes, by definition, are incomplete.

It's important not to fall into the trap of trying to prototype the entire system. I typically pick out 5 or 6 key scenarios I want to focus on at a single time. I'll prototype only what I need to communicate the most important aspects of that concept, things that might not be explicit in the design, or transitions that have some type of wow factor, or impact the user experience.

Todd Zaki Warfel on Prototyping *(continued)*

If I have 2 or 3 really solid concepts I want to explore, then I might prototype the 2 or 3 different solutions and do A/B testing to see which one performs best. Sometimes, I intentionally leave holes in the prototype, to elicit design ideas from participants. When they come across an aspect that isn't fleshed out, I'll ask them "How would you expect that to work?"

What are some of the difficulties with prototyping?

Deciding what pieces to prototype and how deep to go. It's that balancing act of getting the right amount, without doing too much.

The most common mistake is not setting expectations appropriately. If you don't set expectations of the stakeholders you're demoing your prototyping to, then you're going to end up spending 20 minutes defending the parts you left out. If you tell them upfront that this version is going to focus on the drag-and-drop features of the shopping cart, but that address verification isn't included in this round, then your audience will focus on the shopping cart interactions and not get hung up on field verification issues. It's really a bit of selling it appropriately.

What level of fidelity should interaction designers prototype at?

The right level. I know it's a bit of a cop-out answer, but that all really depends on the audience and intent of your prototype. If you're prototyping to show another designer, then something rough and low fidelity might be best. If you're going to demo it to the CEO, then you probably need something a bit more polished.

Be careful about going too polished early on, as you risk customers not giving feedback. Prototypes that are too complete can leave the CEO thinking that all you have to do is ship it, or customers thinking all the design decisions have been made.

How can designers get the most out of prototyping?

Take a page from Nike's book and just do it. I was nervous the first time I started prototyping—frankly, I was probably in way over my head. But I figured it out. The more I prototyped, the easier it got. And now, even when faced with some seemingly impossible designs to execute, I know I'll find a way to figure it out.

The best way to get the most out of prototyping is to just starting doing it. Once you start, you'll never turn back, and you'll wonder how you ever got by without it.

Ideally, designers will create multiple prototypes, or at least multiple variations that can be played with and tested. Designers use prototypes to experiment and see what works—for the designer, for clients, and for users. Frequently, one prototype is clearly the right approach, but just as often, after testing, it becomes clear that parts of each prototype work well, and the designer has to shape these into another prototype that is a hybrid of the best of the earlier prototypes.

Prototypes fall somewhere on a continuum of low- to high-fidelity. The type of prototype you build should depend upon the type of feedback you want to receive. If you want to evaluate overall functionality and product flow, low-fidelity prototypes are appropriate. For more detail on elements such as look and feel and animation, high-fidelity prototypes are more appropriate.

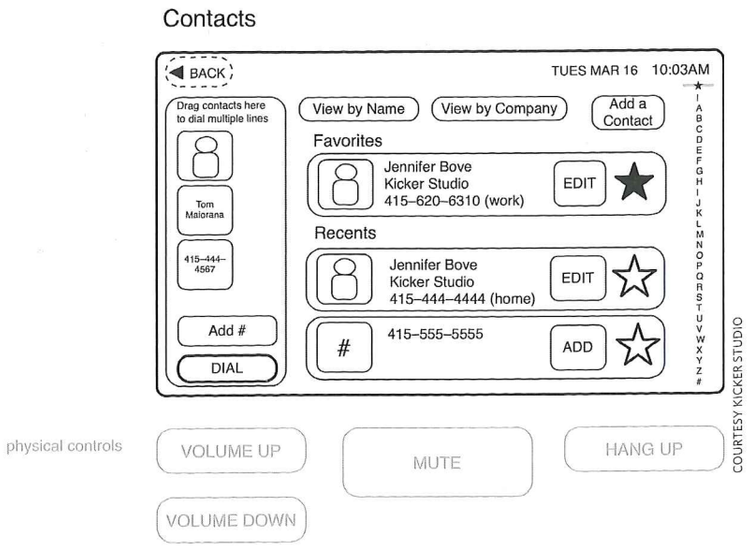
Low-Fidelity Prototypes

Low-fidelity prototypes are put together quickly and are usually crude and unpolished. They might be sketched on paper, made out of cardboard, or perhaps digital but with limited functionality and a basic interface. Low-fidelity prototypes frequently don't "work"—that is, they're usually static with no real interactivity at all. They require people to make them function, by faking any system behavior. Low-fidelity prototypes are meant to be put together (and thrown away) quickly: in just enough time to test a concept.

Since low-fidelity prototypes are not or only somewhat interactive, they usually require someone to be controlling them in order to make them appear interactive. This is called **Wizard of Oz manipulation**, because "the man behind the curtain" (usually the designer) has to make the product seem interactive, either by flipping pages of the paper prototype or by controlling how the digital screen reacts (by flipping to a particular screen when a user pushes a button, for example).

Paper Prototypes

The simplest forms of prototypes are those on paper. Because they aren't digital, some have argued they aren't prototypes at all, but paper prototypes (**Figure 8.1**) do have value for testing the product's overall concept and flow. Indeed, the very fact they aren't digital means that "users" have no expectations that this is the finished product and so are free to comment on them critically.



COURTESY KICKER STUDIO

Figure 8.1

This to-scale, paper prototype of a touchscreen phone was tested with users alongside the physical form.

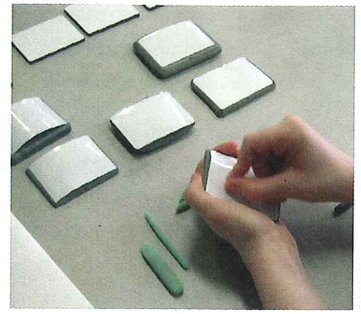
Paper prototypes are usually the fastest way to demonstrate a working product. On paper, the designer creates a walkthrough of the product or system. Each piece of paper contains one moment of the design. That moment can be a Web page, a screen, or a state. Users, aided by the designers, can step through the prototype by flipping through the pages in a particular order. Pages should be numbered, and instructions for moving between the pages should be provided ("If you 'press' this button, go to page 9").

During testing (described in the next section), the subjects or the designer can write comments and notes directly on the prototype ("I really want this button to be *here*"). Even though they are rough, paper prototypes should be done to scale when possible. It is easy to create impractical interfaces otherwise.

Physical Prototypes

Physical prototypes (**Figure 8.2**) can be made for simple parts of a design (such as a dial or a button) or they can be representations of the whole device. Appliances, consumer electronics, control panels, and mobile and medical devices need to have their physical form prototyped alongside the screen whenever possible (using the functional cartography detailed in Chapter 7) simply because there is more to the experience of using a device than just

the screen itself. A device's physical form—especially when it is held in the hand—can drastically change both its behavior and the behaviors of its users.



COURTESY CHUN-YI CHEN

Figure 8.2

These physical prototypes of a small mobile device did variations on size, grip, stylus, and overall shape.

Low-fidelity physical prototypes can be made of almost anything: wood, paper, cardboard, clay, plastic, foamcore…just to name a few materials. For low-fidelity prototypes what is important is the suggestion of form and the location and kind of controls and sensors, as well as size, shape, and weight.

High-Fidelity Prototypes

Once the general concepts, product forms, and task flows have been prototyped using low-fidelity methods, it is time to focus on making a **high-fidelity prototype** (**Figure 8.3**). As the name suggests, these prototypes require a more serious investment in time and resources to create.

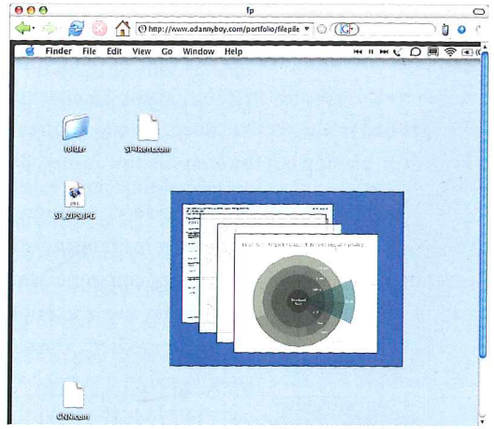


Figure 8.3

Purely digital prototypes like this one can be easily distributed on the Web.

Unlike a low-fidelity prototype, the high-fidelity prototype mostly works as it should. When a user turns a dial, something happens that doesn't require Wizard of Oz trickery to make it work. The data might not be live,

but product mostly behaves as it would in the field, and contains as many of the product details as possible: in interaction, environmental, industrial, and visual design, as well as in engineering and code.

Aesthetics matter when crafting high-fidelity prototypes. Even though it only occurs occasionally, the prototype should be nearly indistinguishable from the product a user would buy or encounter. The less the high-fidelity prototype seems like a prototype, the more accurate the feedback will be. Users shouldn't be confused at being handed a stack of papers and a cardboard box and told to imagine it is an in-store retail kiosk.

For complex functionality, the richer and more complete a designer can make the high-fidelity prototype, the better. It is hard for users to imagine how, for example, a tool to draw boxes might work without actually being able to play with it and draw boxes themselves.

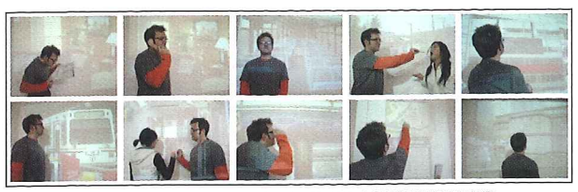
The danger with a high-fidelity prototype is that both users and clients may think it is the final product. Expectations should be properly set for what the prototype actually is: a prototype.

Service Prototypes

Prototyping a service usually isn't much like prototyping a product. Since both the process and people are so important to services, services don't really come alive until people are using the service and walking through the process. Prototyping a service typically involves creating scenarios based on the service moments outlined in the service blueprint and acting them out with clients and stakeholders, playing out the scenarios as theatre. Seriously.

Role playing constitutes a significant part of the service design process. Only through enactments can designers really determine how the service will feel. Someone (often the designer) is cast in the role of employee, while others play the roles of customers. This prototyping can make use of a script or an outline of a script, or the enactments can simply be improvised. The players act their way through a service string to demonstrate how the service works.

Ideally, these scenarios will be played within a mock-up of the environment (**Figure 8.4**), with prototypes of the objects involved as well. Only in this way can the flow and feeling of the service really be known. Environments can be simulated using giant foam blocks for objects, masking tape on the floor to block out areas, images projected on walls, and so on.



COURTESY ANNIE HA, ROSEMARY LAPKA, JEEWON LEE, AND PURIN PHANICHPHANT

Figure 8.4
A prototype of a subway service, created by projecting images behind the designers/actors.

One alternative ("Wikitecture") some designers and architects are experimenting with for prototyping spaces is to model environments in a virtual world such as Second Life.[1]

Services also can be prototyped using a live environment with real customers and employees. The Mayo Clinic's SPARC program does this (see the case study that follows), as do many retail stores, using pilot programs at a small number of locations. These prototypes are, of course, extremely high fidelity, working exactly as the actual service would because they involve actual customers. Although it is certainly best to start with low-fidelity service prototypes (if only because of the cost), eventually the service will need to be tested with actual customers and employees, either in a prototype/pilot environment or live, making adjustments as the service lives.

Testing

Once you have a prototype, the product or service should be tested with users. This process is usually called user testing, but that's really a misnomer; it's the product or service that's being tested, not the users (**Figure 8.5**).

The same rules that guide design research (see Chapter 4) also guide testing: you go to the users, you talk to them, you write things down. Unless what is being tested is a service that requires a prototyped space or some other significant setup, testing is best done in the subject's own environment: on the subject's computer, in the subject's home or office, in the subject's city or town.

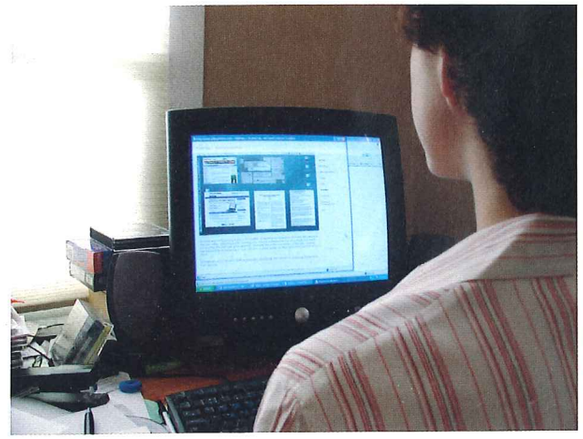


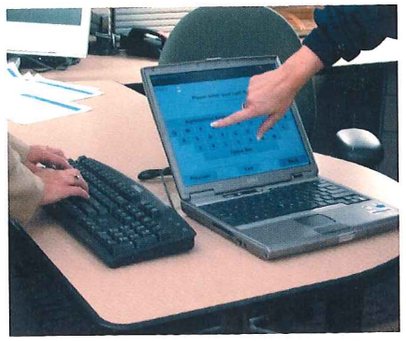
Figure 8.5
A digital prototype being tested by a user at home.

1 See The Arch, a blog by Jon Brouchoud at http://archsl.wordpress.com/

Case Study: Electronic Check-In System

The Company

The Mayo Clinic, an internationally-known medical facility.

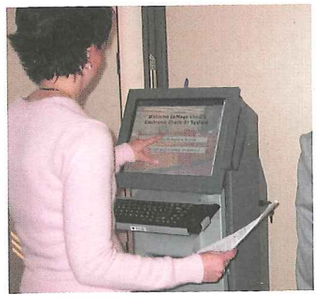


The Problem

Even though the delivery of health care hasn't changed much in 50 years, most patient satisfaction about health care comes through the delivery of that care, not necessarily the care's effectiveness. Designers at the Mayo Clinic observed that a point of patient annoyance is checking-in simply to say that they had arrived. The check-in process sometimes even exacerbates medical conditions.

The Process

The Mayo Clinic's SPARC (See, Plan, Act, Refine, Communicate) program was created for just such a situation as the check-in process. SPARC provides live-environment (real patients, real doctors) exploration and experimentation to support the design and development of innovations in health-care delivery. SPARC is both a physical space and a methodology combining design and scientific rigor. Embedded within a clinical practice in the hospital, the SPARC space has modular furniture and movable walls that allow many different configurations, and it is staffed with a blend of physicians, business professionals, and designers. Using the airline industry as a model, SPARC designed a prototype of a check-in kiosk, collected initial feedback from potential users, and then iteratively refined that prototype.



The Solution

SPARC designed a self-check-in service similar to airline check-in services at airports. Patients check in using a kiosk instead of waiting in a line just to say that they have arrived. SPARC tested the kiosk with 100 patients and found a high rate of acceptance and significant reduction in the number of interactions required while the patient is waiting for service. There was also a marked reduction in patients' waiting time.

Testing labs do have two advantages: efficiency and a controlled environment. The designer can quickly see many subjects in a single day, one after the other, without having to change location, and there is only one setup.

Also similarly to design research, subjects need to be recruited, and a moderator script (also known in this stage as a **test plan** or **testing protocol**) devised. In the test plan, a "route" through the product that tests the functionality and feedback is devised, as well as questions that would prompt (although not guide) users through the system. Strive for neutral questions such as, "If you wanted to check your account balance, what would you do?"

Test plans also have to take into account the limitations of the prototype. Often, prototypes do not use live data, so only certain kinds of input will work properly, and often only certain pathways through the system have been built. For example, to get to a particular page, users will have to search for a particular name. Other names won't work, so the moderator may have to prompt the subject to enter that particular name.

A/B testing (sometimes called "bucket testing") is a particular method of testing wherein two different designs are shown to users, and then the results compared to see if one is markedly better than the other.

Testing is also the time when any wrong conclusions reached during design research can be corrected. Designers may find that they misinterpreted the research or drew the wrong implications from the research. By talking to users during testing, they can clear up those misconceptions. Ideally, designers will carry a set of wireframes and other documents during testing and make notes right there about any patterns they see. For example, if users keep stumbling while attempting to complete a registration form, the designer should plan to revise that form later.

Designers, when testing, should not be defensive about their designs; indeed, it is often best for the designer to allow other team members or usability specialists to conduct the testing while they simply observe and take notes. The human tendency is to nudge subjects ("Why don't you just look over there at that button?") because the designer knows the design better than the subject. To avoid inhibiting testers, designers should avoid identifying themselves as the product's designer. Knowing that the designer is present may cause testers to change or soften their feedback.

Test results can be delivered in an **opportunity report**, which indicates where in the product users had trouble, and also possible suggestions for improvement based on test results and/or subject suggestions.

Most experienced designers know one truism: you seldom get it right the first time. Testing will reveal the flaws, both known and unknown. Nothing is more humbling for designers than watching users stumble through their designs. While testing, good designers watch for the same thing they watched for in design research: patterns. Then they go back to the wireframes and prototypes and fix the errors. Then they test them again. This is how products and services should be made.

Heuristic Evaluation

If you don't have the resources to do testing with users, the least you can do is perform a heuristic evaluation on the prototype yourself. Walk through the prototype as though you hadn't designed it and didn't already know everything about it and why features are the way they are. Look for the following things:

- **There are too many actions, clicks, or steps to do key features.** This indicates that important functionality is buried or inefficient. Consider redoing the framework or task flows.
- **Lack of explanation.** If you wonder why you are performing a task, it's guaranteed users will wonder, too. Either information (a label, a description, a process indicator such as "You are on Step 3 of 4") needs to be provided, or the feature needs to be rethought from a strategy or framework perspective—for example, do you really need this feature?
- **Huh? What just happened?** If you don't understand the result of an action, the feedback and/or feedforward are likely poor.
- **Did anything just happen?** If you or the system performs an action and you can't tell, the feedback is inadequate.
- **Hidden features.** Are there features that are difficult or impossible to find? Are there functions hidden in modes that should be surfaced to the top? Review information architecture/frameworks.

- **Lost.** If you are somewhere in the system and can't figure out where you are or how to get back, it's an information architecture/navigation problem.
- **Where's my data?** We expect a system to remember basic things about us, and especially things that we have taken the time to tell it (user information, settings, and so on). When data isn't there, it can cause anger, frustration, and concern.
- **If I click this, what happens?** If you can't tell what is going to happen when you press a submit button or flip a switch, the feedforward/label is bad. This can also speak to a lack of understanding of a feature's purpose.
- **I didn't see that button.** If a key control isn't visible, the layout, visual hierarchy, or affordances are poor.
- **Dead ends.** Error messages, becoming trapped in a feature or mode, or being unable to undo an action are signs that the task flow could be bad.

All of these situations can trip users up and should be avoided.

Development

The final step before the product or project is launched is the actual development—and, for physical objects, manufacturing. Once in this stage, the designer's role (unless he is also the developer) is one of troubleshooting, tweaking the design to fit the code and/or the physical materials, and collaboration.

Essentially, the designer should be part of the development process in order to ensure that the product comes out as envisioned. Once the building process has begun, assuredly issues will arise that had not been thought of, and designers should make themselves available to work through those issues alongside the developers.

Leisa Reichelt on Designing Throughout the Development Process

Leisa Reichelt is a contextual researcher and user-centered designer who has been designing for over a decade, with her most recent and public project being Drupal 7. She also coined the term "ambient intimacy" to describe that sense of connectedness that you get from participating in social tools online that allow you to feel as though you are maintaining and, perhaps, in fact, increasing your closeness with people in your social network.

Why should designers bother with being involved in the development process?

Firstly, the design process is the development process and the development process is the design process. The idea that they are separate from each other is a tragic misconception.

Design decisions happen well beyond the end of what we'd traditionally recognize as the "design phase" of a project—if you want to be a part of this ongoing design decision-making then you need to be there when the decisions are being made.

This happens when developers interpret your specifications, when they assume you must have meant something different because what you've done looks nuts to them, when they find something that hasn't been specified at all and make it work how they think it should work. It's not a failing in your specification that these things happen, it's just the way it works. You can acknowledge this and get involved or you can forever wonder why things never leave the developers looking and working like you want them to...your call.

Secondly, it's about knowing your media. You'd be horrified if you asked an architect to design you a house and she knew nothing about the materials she was specifying. Similarly, you should know about the materials that you're using in your design—this includes the code. You don't need to know exactly how to code up your designs but by working closely with developers you learn about the capabilities, potential, and restrictions of your media. This can only help you be a better designer.

There is a lot of talk about how design fits into the Agile development process. How do you see that?

Unfortunately it is often a whole world of pain. Agile is a really great idea in many ways but it was born of developers and, unfortunately, the vast majority of Agile projects are still very much

Leisa Reichelt on Designing Throughout the Development Process *(continued)*

development-led. This means that there is very little room for designers to work in an Agile way and many developers are actually really uncomfortable working with designers who are also working Agile and therefore don't know the whole story of what we're building yet either!

There is a lot of great work being done, and a lot still to happen, to try to get design to work better in an Agile environment, but it is still very much a work in progress. What we do know is that it is all but impossible to do good design work whilst trying to fit into a traditional strict Agile method—we as designers need to engage more in trying to shape the methodologies so that they better suit our work practices, and then talk to each other about what works and what doesn't so we can start to develop a better shared understanding and get a design-friendly version of Agile out there!

What's the "washing machine" development process you advocate?

It's a kind of silly term with a sketchy diagram that I use to describe the way that we need to mess with Agile so that design fits into it better.

There are many great things about Agile that should support good design—the emphasis on prototyping (creating working releases), incremental work, working to user stories, working collaboratively, short bursts of high focus—all of these should contribute to good design outcomes. There are other aspects of good design practice that need to be integrated better into Agile though. For example, involving real end users rather than just end user representatives, allowing designers a period of time (that we're now calling Iteration Zero) to conduct research, to come up with an overall design framework, and to allow for more iterative work—you don't see enough iteration in most Agile projects.

What's the best way of seeing your design make its way through the development process and end up working like you'd imagined it?

I think perhaps to spend less time imagining it and more time being there, making it happen. What I'm really aiming to do these days is as little upfront documentation as possible, but rather to work closely with the developers as early as possible to start getting things built—and with the time that I would have otherwise spent annotating wireframes, I get to work with the developers to fine tune the design and interaction, to do more iteration. I'm not sure if it's just that I do less imagining at the beginning so I get less disappointed at the end! I don't think so. I think it's just a much more satisfactory way to work.

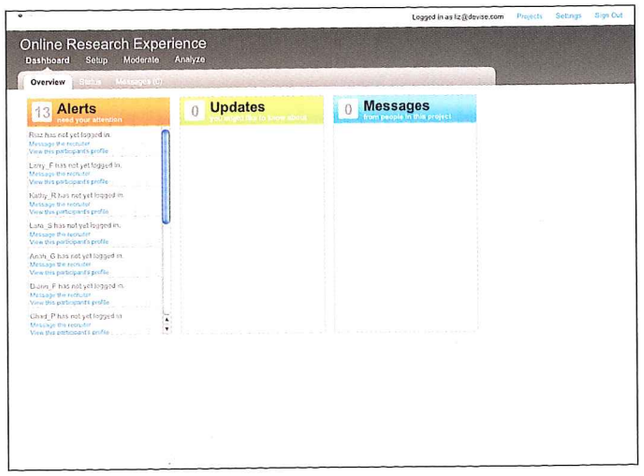
Case Study: Revelation PROJECT

The Company

Revelation, a Web application startup that provides software and services for qualitative researchers.

The Problem

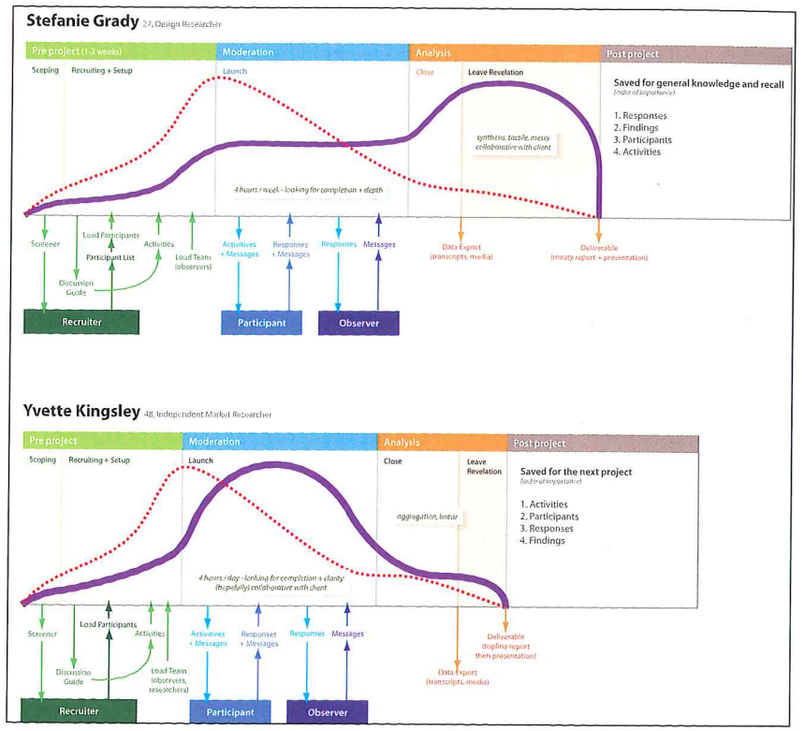
Revelation's product PROJECT, a Web application for researchers to set up daily diaries, photo essays, questionnaires, and other creative stimuli for their study participants, in its first iteration had some significant usability and performance problems that were causing issues for their users. Revelation wanted a new version of PROJECT before a major industry event, so rapid results were required.



The Process

Working with design firm Devise, Revelation did ethnographic research in two locations with 10 researchers who conducted a mix of market research and design research. The team ultimately created six personas from their research: a market researcher, a design researcher, a client observer (the person requesting and paying the researcher to do the work who gets to do certain things in the system), two study participants, and the recruiter (the person who would find and put participants into PROJECT on behalf of the researcher). The personas, placed into scenarios, drove the new design.

Case Study: Revelation PROJECT *(continued)*

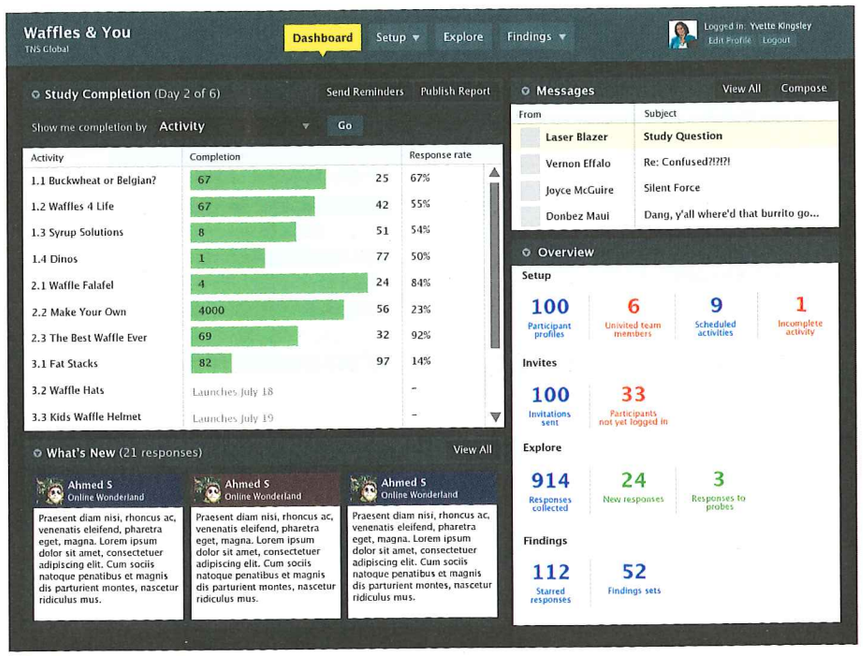


The project had a tight timeline, beginning in June with a release date of September 30th. The development team had successfully implemented an Agile Extreme Programming (XP) process. The design team worked closely with developers, integrating their design work with that methodology. The personas' usage scenarios allowed developers to quickly learn and understand customers' needs and goals. Given the short time frame, decisions needed to be made on which features should be given priority. The team decided that the biggest initial business opportunity resided with market researchers versus design researchers, and so features that spoke to the market researcher persona were prioritized for the initial release. Key path scenarios for the primary persona became Agile's "story cards," allowing the design and development team to collaborate closely and communicate clearly. Features that were more focused toward design researchers were placed lower in the story queue for subsequent development iterations.

Case Study: Revelation PROJECT *(continued)*

The Solution

In four weeks' time, the designers made significant improvements to the product. The Dashboard home page became more action-oriented to better help researchers moderate activities and analyze data. Setting up a research study was streamlined. Researchers are now able to moderate and interact with participants' responses using natural-language statements such as: "Show me what's new" or "Show me all responses tagged with..." or "Show me all responses from this participant...."



The design team designed multiple analysis tools for researchers and their clients to approach participants' responses from many different angles. PROJECT users can dynamically build data sets around any theme or idea. Research showed that improving collaboration between researchers and their clients was an important opportunity area, and the simpler analysis tools let clients get involved without any training overhead.

Agile

Agile is a particular programming methodology that, as of this writing (2009), is deeply in vogue. It arose in the early 2000s as a reaction to what is known as "Waterfall" methods, wherein developers are handed a large stack of documentation ("functional specifications") and told to build them. (Truthfully, much of the process outlined in this book could be seen as a waterfall-like process.)

The essence of Agile methodology is the breaking of larger tasks/features into small pieces to be built in short bursts of development that typically last from one to four weeks. A small team works on these "iterations" together through the full development cycle: planning, requirements analysis, design, coding, unit testing, and acceptance testing, wherein they demonstrate the working piece to stakeholders. Agile helps minimize overall risk, and more easily allows for changes to the functionality.

What is challenging for designers working in this methodology is that it doesn't allow for the big picture—strategic and frameworks—thinking that designers need to engage in, nor does it seem to provide enough time for ideation and exploring multiple options. An ideal situation is to allow for a more traditional design process (such as outlined in this book) to occur, but then turn to Agile methods (with a designer embedded in the team in order to make changes to the design as necessitated by the code) for development.

Summary

Prototyping, testing, and development are the final crucial steps in the design process, where all the strategy, research, ideation, design principles, and refinement come into their full bloom and the product comes alive. It's important for designers to not abdicate responsibility for the final outcome of their designs to those who build them, for the simple reason that no matter how complete you think your documentation is, developers and manufacturers likely do not have all the information and the product vision you possess.

It's also important to note the "end" of the design process is seldom the end. Products, even after launch, are always evolving and take on a life of their own as users begin to use them in their daily lives. Problems and opportunities will arise, and the market will change. And then the process starts all over again.

For Further Reading

Designing Interfaces, Jenifer Tidwell

Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces, Carolyn Snyder

Making Things Talk, Tom Igoe

A Practitioner's Guide to Prototyping, Todd Zaki Warfel

A Practical Guide to Usability Testing, Joseph S. Dumas, Janice C. Redish

The Art of Agile Development, James Shore and Shane Warden

Designing the Moment: Web Interface Design Concepts in Action, Robert Hoekman Jr.

GUI Bloopers 2.0: Common User Interface Design Don'ts and Dos, Jeff Johnson

Don't Make Me Think: A Common Sense Approach to Web Usability, 2nd Edition, Steve Krug

9

The Future of Interaction Design