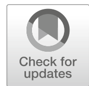

Participatory Design and Prototyping

Wendy E. Mackay and Michel Beaudouin-Lafon

Contents

Introduction 2
Participatory Design 3
Prototypes Help Explore the Design Space 5
A Taxonomy of Prototypes 8
Representation 9
Precision 11
Interactivity 13
Life Cycle 16
Scope 17
Summary 18
Methods and Tools for Rapid Prototyping 19
Physical Rapid Prototyping Methods 19
Rapid Nonfunctional Software Prototyping Methods 24
Tools for Developing Functional Prototypes 26
Summary 30
Summary 30
References 32

Abstract

Participatory design actively involves users throughout the design process, from initial discovery of their needs to the final assessment of the system. As in all forms of interaction design, this requires two complementary processes:

W. E. Mackay (✉)
Inria and Université Paris-Saclay, Orsay, France
e-mail: wendy.mackay@inria.fr; Wendy.Mackay@lisn.fr

M. Beaudouin-Lafon
Université Paris-Saclay, Orsay, France
e-mail: michel.beaudouin-lafon@universite-paris-saclay.fr; mbl@lisn.fr

© Springer Nature Switzerland AG 2023

J. Vanderdonckt et al. (eds.), *Handbook of Human Computer Interaction*,
https://doi.org/10.1007/978-3-319-27648-9_31-1

generating new ideas that expand the design space and selecting specific ideas, thus contracting the design space. Interaction designers create prototypes that help them generate and explore this space of possibilities, ideally with the ongoing participation of users. Successful prototypes represent different aspects of the system that highlight specific design questions. We classify prototypes based on: *representation*, which refers to their physical form, ranging from paper, cardboard, or foam mock-ups to complete software or hardware simulations; *precision*, which refers to the level of detail, ranging from simple hand-drawn sketches or physical mock-ups to polished computer-generated images or 3D printed objects; *interactivity*, which refers to the level of interaction possible, ranging from no interaction, such as a video scenario, to partial interaction, when a designer "walks through" a scenario, to fully interactive, either simulating interaction with the Wizard of Oz method or trying an operational software or hardware prototype; *life cycle*, which refers to the expected evolution of the prototype, from "throw-away" prototypes in the earliest design phase to successively more developed prototypes in later design phases, to actual components of the final system; and *scope*, which refers to the aspects of the final system covered by the prototype, including breadth-first horizontal prototypes, depth-first vertical prototypes, or path-based story prototypes. This chapter explains how to create and use these different kinds of prototypes, with examples selected from key phases of the interaction design process.

Keywords

Prototyping · Participatory design · User-centered design · Rapid prototyping · Paper prototyping · Video prototyping · Software prototyping · Taxonomy · Prototype representation · Prototype precision · Prototype interactivity · Prototype life cycle · Prototype scope · Wizard of Oz · Wireframe · Mock-up · Physical prototype · Functional prototype · Software prototype · Fidelity

Introduction

Prototypes serve as an essential design resource for interaction designers, especially within a user-oriented participatory design process. Because it is impossible to consider all design possibilities at once, designers create prototypes to help them assess specific design alternatives. Prototypes also help designers articulate ideas and communicate them to other design team members, as well as users, clients, management, and other stakeholders.

This chapter first defines participatory design, also called co-design, and explains its role as a key form of user-centered design. We then introduce the concept of a design space, and how participatory design involves continually transitioning between gathering new information that enlarges the design space and making specific design choices that reduce the design space. We show how prototypes can

both inspire new possibilities, but also enable comparison of details that clarify specific design choices.

We next define what we mean by a “prototype” and introduce a taxonomy for describing, analyzing, and choosing prototypes with the appropriate focus, at the appropriate level of detail, for a given phase of the design process. We conclude with a range of specific examples, from early-stage rapid prototyping to more highly refined, computer-based prototypes more appropriate later in the process.

Participatory Design

Interaction design should be both user-centered – the user is the focus of the design – and iterative, design concepts are successively regenerated and revised (Norman and Draper 1986). However, in practice, user-centered design is often limited to questioning users to get a basic idea of their needs (“need-finding”) at the beginning of the design process, and then performing user studies to evaluate the resulting system at the end of the process.

Participatory design (Schuler and Namioka 1993; **?**) is a special case of user-centered design, with the specific goal of actively involving users throughout the design process, from early identification of user needs to final assessment of the system. Originally called “co-operative design” (Greenbaum and Kyng 1991), the approach was explored extensively in Scandinavia in the 1960s and 1970s to empower users and give them a voice in technology design. Within the field of human-computer interaction (HCI), Muller and Druin (2002, p. 3) define participatory design as “a set of theories, practices, and studies related to end-users as full participants in activities leading to software and hardware computer products and computer-based activities.” Over the past 30 years, participatory design has become increasingly popular and includes a diverse set of practices that support what Bødker and Buur (2002) call the “many-voiced nature of design.” Today, the term “co-design” is sometimes used as a synonym for participatory design, especially within the user experience (UX) design community.

Early and active involvement of users helps designers develop a deeper understanding of the actual design problem and avoid faulty design paths. Obtaining user input at each phase also changes the nature of the final system evaluation, changing the focus to fine-tuning the interface rather than discovering major usability problems. Although it may sound expensive, especially if gaining access to “real” users is difficult, participatory design is often significantly cheaper in the long run. By refocusing the design on actual user needs, participatory design helps avoid “minor” design errors that lead to catastrophic failures and can even benefit from unexpected user innovations.

One common misconception about participatory design is that designers are expected to abdicate their responsibilities as designers and leave the design to users. This is never the case: designers must always consider what users can and cannot contribute. Normally, users’ key contributions lie in their deep understanding of the subject matter and the context of use, as well as the specific needs that arise within

that context. A few individuals may also generate "user innovations" or solutions to a common problem that would benefit other users in similar situations. Although innovative ideas may arise from both users and designers, it is the design team who must judge each idea's feasibility. Because they must design and implement the final system, designers and developers are responsible for considering the range of options and constraints and for balancing the trade-offs among them.

One of the challenges of conducting true participatory design is how to successfully involve users in each phase of the design process. Muller and Druin (2002) describe how low-tech prototypes bring users into new relationships with technologies, not only to think about technologies or applications they have not previously experienced but also to use low-tech materials to reshape those technologies. Because prototypes are shareable, physical artifacts, they serve as an effective communication medium for users and the design team. Collaborating on prototype design is an effective way of involving users, since they provide a medium for users to articulate their needs and reflect upon the efficacy of proposed design solutions.

We contrast our approach to that of Lim et al. (2008), who describe prototypes in the context of a fundamentally *unidirectional* process where designers, ranging from architects to interaction designers, use prototypes to filter the design space and concentrate on a few key design issues at a time. We focus instead on interaction designers who use prototypes in a fundamentally *bidirectional* process as a means of engaging users throughout the design process, as they explore how users will interact with the yet-to-be-developed system.

Although prototypes are useful in all forms of user-centered design, they offer a particularly effective method of involving users as full participants in the specification and evaluation of the design process. Users can help construct personas and define how they would interact with both existing and future technologies. Interaction designers may also use prototypes to communicate with users and enable them to imagine, experience, and influence diverse aspects of the final system, long before it is built. Prototypes play an important role in bringing these stories to life: not only to identify interesting new design directions, but equally important, to highlight key breakdowns and potential workarounds when the technology is used in a realistic setting. Users can also create their own prototypes, especially paper mock-ups or quick video scenarios (Mackay 2020) (Fig. 1), to generate new ideas and explore interesting alternatives.

Every participatory design process incorporates a variety of different types of prototypes, created by both designers and users, at all stages of the design process. Each prototype reveals the strengths and weaknesses of a specific design option and can be contextualized to illustrate how users would interact with the final system in a real-world setting. Prototypes clarify functional requirements and identify potential usability and performance issues. Because they are concrete and detailed, prototypes let designers explore and compare realistic scenarios that users can assess relative to their actual needs. Designers can also compare prototypes to other existing systems and use them to better understand users' work practices and context of use. Each prototype, whether a quick paper prototype or an elaborate functional system, lets both designers and users to interpret and reassess users' needs.

Fig. 1 Designers and users collaborate on creating a physical prototype in a participatory design session. Here, participants are creating a simulated screen with movable sticky notes that represent pop-up menus and windows, which will later become the basis for a video prototype

Prototypes Help Explore the Design Space

The goal of interaction design is to create something new that meets the needs of users. Although designers are not scientists, they often borrow findings and methods from both the natural and social sciences (Mackay 1988). However, designers also require additional design methods that help them generate new ideas and balance complex sets of trade-offs as they develop and refine those ideas.

Paul Laseau (1980) describes design as an ongoing process that involves idea *elaboration*, as designers search for new opportunities, and idea *reduction*, as they make specific design choices. In 2005, the British Design Council (2005) presented this in the form of a *Double Diamond*, a simplified visual representation of the design process, to capture the alternation between exploring an issue more widely (divergent thinking) and taking focused action (convergent thinking). Designers use the concept of a *design space* to articulate the set of possible design ideas and constraints that result from divergent thinking. Exploring the design space allows designers to reject certain ideas and leave others open for creative exploration.

A design space comprises multiple dimensions and individual ideas can be compared based on their place within each dimension. Designers use many different strategies for representing a design space, including tables, two- or three-dimensional graphs, and RADAR diagrams, such as the one shown in Fig. 2. This sketch shows six dimensions relating to the design of "Communication Apps," including number of participants (two to three people, small group, big group), type of information (audio, images, video), what participants see (symmetric or asymmetric), continuity (discrete, mixed or continuous), and what information is captured (drawing, typing, shooting images, or detecting biometric information).

Designers can use colors to highlight different design options within this design space. Here, the blue line represents a screen-based image exchange system, modeled after the "Video Probe" idea (Hutchinson et al. 2003). The yellow line represents an idea for a live audio heartbeat shared between two people. The orange line represents a shared video recording and collaborative editing system for a small group. As the details of a particular design are narrowed down, many of

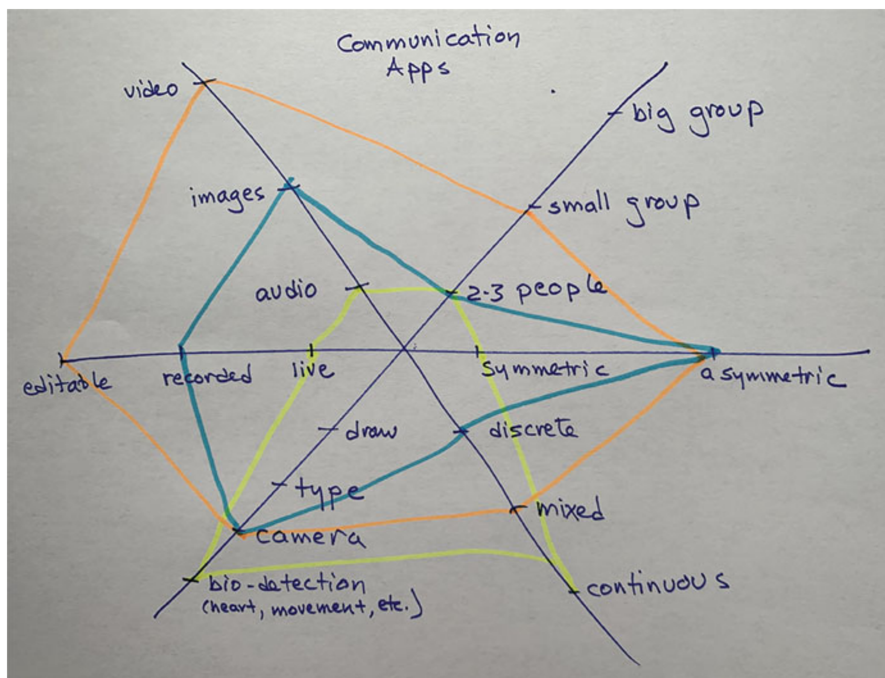

Fig. 2 A quick, hand-drawn design space where each dimension includes multiple options. Different design alternatives are shown in different colors

these dimensions become fixed, and the designer considers additional, usually more detailed, design dimensions. For example, the designers might decide that sharing a heartbeat is too intrusive and identify other biometric information that might be shared between members of a small group of workout enthusiasts.

The designer can expand the design space by adding more options, more dimensions or identify gaps at the intersections of two or more dimensions. For example, the Octopocus dynamic guide (Bau and Mackay 2008) uses progressive feedforward to show which gesture commands are available in a gesture-based interface. The design space shown in Fig. 3 compares existing systems according to their level of detail and update rate and shows a major gap that Octopocus fills, since the guide is continuously updated as the user draws the gesture.

Design space ideas come from many sources: existing designs, other designers, external inspiration, or even accidents that prompt new ideas. Designers are responsible for creating a design space that is specific to a particular design problem. They explore this design space, expanding and contracting it as they add and eliminate ideas. The process is iterative, more cyclic than reductionist. That is, the designer does not begin with a rough idea and successively add more precise details until the final solution is reached. Instead, she begins with a design problem, which imposes a set of constraints, and generates a set of ideas to form the initial

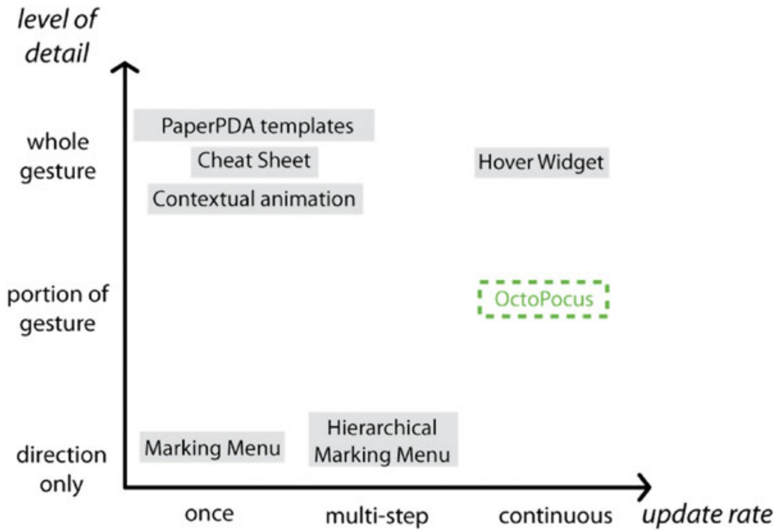

Fig. 3 The design space comparing Octopocus (green) with other techniques for providing users with in-context help, with *update rate* on the X-axis, and *level of detail* on the Y-axis

design space. She then explores this design space, preferably with users, and selects a particular design direction to pursue. Each exploration closes off part of the design space, but opens up new possibilities to be explored. The designer can expand the design space by adding new options along any particular dimension, or adding entirely new dimensions. She can then explore the expanded design space, paying special attention to gaps that appear or interesting new intersections, and make new design choices.

All designers work with constraints: not just limited budgets and programming resources, but also design constraints. Working with constraints is actually important, since a designer cannot be creative along all dimensions at once. However, some constraints are unnecessary, derived from poor framing of the original design problem. If we consider a design space as a set of ideas and a set of constraints, the designer has two options. She can modify ideas within the specified constraints or modify the constraints to enable new sets of ideas.

The ability to consider different kinds of constraints is correlated with different kinds of education. For example, engineering students are trained to treat the design problem as a given, whereas design students are encouraged to challenge and, if necessary, change the initial design problem. If the designer reaches an impasse, she can either generate new ideas or redefine the problem (and thus change the constraints). Some of the most effective design solutions derive from a more careful understanding and re-framing of the design brief.

Note that all members of the design team, including users, may contribute ideas to the design space and help select design directions from within it. However, the activities of expanding and contracting the design space are normally kept separate

from each other. Expanding the design space requires creativity and openness to new ideas. During this phase, participants should avoid criticizing ideas and concentrate on generating as many as possible. Clever ideas, half-finished ideas, "silly" ideas, and impractical ideas – all contribute to the richness of the design space and improve the quality of the final solution.

By contrast, contracting the design space requires critical evaluation of ideas. Designers and users should work together to consider constraints and weigh the trade-offs. Each major design decision must eliminate part of the design space – rejecting ideas is necessary in order to experiment and refine others and make progress in the design process. Choosing a particular design direction should spark new sets of ideas, and those new ideas are likely to pose new design problems. In summary, exploring a design space is the process of moving back and forth between creativity and choice. Note that this shift between expanding the design space and making specific choices can occur at any time, not only when shifting from one design phase to another, but even within a specific design activity.

One of the best ways to explore the design space is by creating prototypes, which act as concrete representations of new ideas and clarify specific design directions. The next two sections describe the types of prototypes that have proven most useful in research and product development.

A Taxonomy of Prototypes

A prototype is a *concrete representation* of part or all of an interactive system – a tangible artifact, not an abstract description that requires interpretation. A successful prototype highlights essential features that should be investigated at a particular moment in the design process. Various stakeholders, including designers, managers, developers, clients, and users, can all engage with prototypes to explore different design possibilities and to envision and reflect upon the final system.

Prototypes are both artifacts in their own right and important components of the design process. When viewed as artifacts, successful prototypes support *creativity* by helping the designer capture and generate ideas, facilitate the exploration of the design space, and uncover relevant information about users and their work practices. Prototypes also encourage *communication* by providing a common ground for designers, users, and other stakeholders to interact with each other, generate ideas, and choose among design alternatives. Prototypes support *early assessment*, since they can be evaluated in various ways, from informal user feedback to traditional user studies, at each phase of the design process.

Bill (Buxton 2007) argues that sketching is a critical design activity, essential both for "getting the design right and the right design." The act of sketching or building prototypes helps designers to think, whether solving specific design problems or generating new ideas. Prototypes may uncover issues or help designers refine their ideas. They also let designers compare options or choose among alternatives. Lim et al. (2008) describe prototypes as "filters" that highlight certain aspects of the design problem and minimize others. In each case, the prototype acts

as an important resource, integral to the design process. Design is fundamentally about making choices, not only among different design solutions, but about what type of prototype is most appropriate when, as well as how to address a particular design issue. We classify prototypes along five primary dimensions:

- **Representation** describes the physical form of the prototype, from rough sketches to wireframes to full computer simulations.
- **Precision** describes the level of detail, from rough and informal to highly polished.
- **Interactivity** describes the level of interaction with the prototype, from pre-recorded video or animations to fully interactive.
- **Life cycle** describes the expected evolution of the prototype, from rapid “throw-away” prototypes to components of the final system.
- **Scope** refers to the part of the final system that is covered by the prototype, including breadth-first horizontal prototypes, depth-first vertical prototypes and path-based story prototypes.

Representation

Prototypes serve different purposes and thus take different forms. A series of quick paper sketches can be considered a prototype, so can a detailed computer simulation. Both are useful: each supports the design process in different ways. We distinguish among three forms of representation: *physical* prototypes made of paper or other materials, *nonfunctional* software prototypes that illustrate visual or dynamic elements, and *functional* software or hardware prototypes that actually work.

- *Physical prototypes*, often referred to as paper prototypes, include paper sketches, illustrated storyboards, and paper, cardboard, or foam mock-ups (Fig. 4) and may be hand-drawn or printed from a computer. Their most salient characteristic is their rapid creation, usually in the early stages of design, and they are often discarded when they have served their purpose.
- *Nonfunctional prototypes* include wireframes, animations, and interactive video presentations that are created with computer design tools, as well as three-dimensional user interface mock-ups created with computer-aided design (CAD) software and laser cutters or 3D printers. Compared to physical prototypes, these nonfunctional prototypes usually provide more detailed representations of each idea.
- *Functional prototypes* include either working software programs or working hardware that implement part or all of the design, so that users can interact with them as if they were the real system. They can be programmed using a variety of tools, including scripting languages, user interface frameworks and libraries and user interface builders.

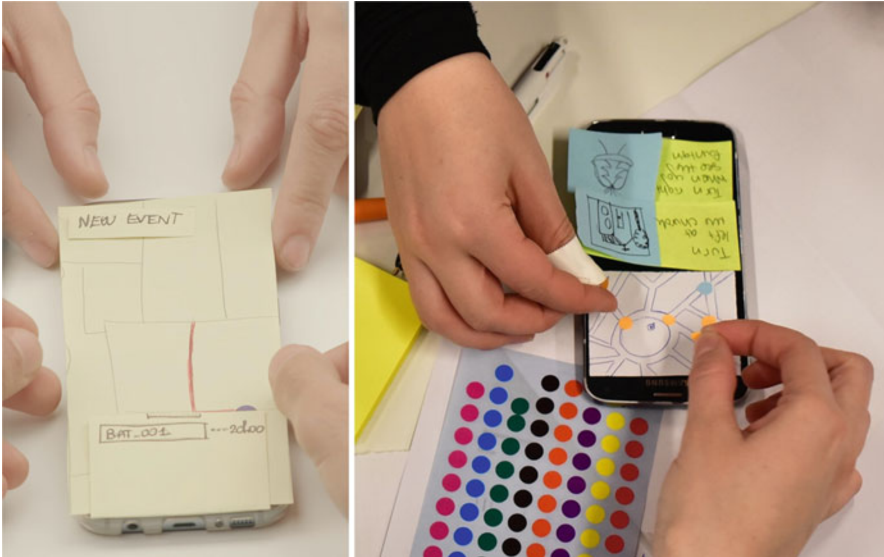

Fig. 4 Left: a physical prototype composed of multiple sticky notes. Right: a paper prototype layered onto a smartphone, with movable elements. The designers move these elements to create the illusion of interacting with the system

Designers of websites and other screen-based systems usually develop *wire-frames* – skeletal drawings of each screen – to show what the user will see on the screen. They usually begin with rough, hand-drawn sketches that illustrate the basic layout, as in Fig. 5:left, and then use more advanced tools to create screens that look like the final system. Each set of wireframes focuses on the key elements of interest at the time they are drawn.

Unfortunately, designers with extensive experience using specific design tools, such as FIGMA or Adobe XD, often prefer beginning the design process with nonfunctional software prototypes. Programmers often make the same argument in favor of coding functional prototypes, even at the earliest stages of design. In both cases, they feel that, given their skill with the tools, they will be able to skip early ideation and quickly move directly to more detailed, precise designs.

In our experience, this is *never* a good strategy, since using a computer-based tool too early almost always slows the process down, rather than speeding it up. Not only are physical prototypes quick and inexpensive, which encourages rapid iteration and more thorough exploration of the design space, but they also help prevent the design team from becoming overly attached to the first reasonable solution.

We argue that designers should begin with physical prototypes (Fig. 5:left) to explore their ideas, and only use software prototypes, whether nonfunctional or functional, to clarify and refine the design (Fig. 5:right). Rapidly created physical prototypes are also far less likely to constrain how designers think or encourage them to commit to a particular idea too soon. Every design tool, programming

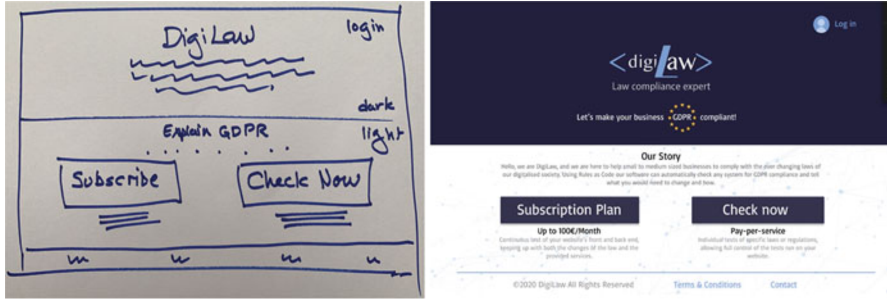

Fig. 5 Left: rough, low-fidelity hand-drawn wireframe. Right: precise, high-fidelity FIGMA wireframe created later in the design process

language or development environment involves a set of underlying assumptions that may or may not be appropriate for the current design problem. They also require significantly more effort and making changes is far more time consuming. In addition, the software libraries that come with each design tool include a set of predefined components which encourages designers to ignore the details of the interaction and instead focus on assembling wireframes, often with preexisting "frames" or "views," instead of creating novel alternatives that meet the user's specific design needs. This imposes specific constraints on both the graphics and the interaction, limits creativity, and restricts the number of ideas that will be considered.

The cost of producing fully functional software or hardware prototypes is even higher. Skilled programmers or engineers will be required to implement advanced interaction or visualization techniques or to meet tight performance constraints. Functional prototypes should thus be created during the later design phases, when the basic design strategy has been decided.

Finally, and perhaps most importantly, physical prototypes can be created by a wide range of people, not just programmers, and act as a critical component of any true participatory design process. Everyone, including designers, programmers, managers, clients and users, can all contribute on an equal basis. Unlike using a design application or programming language, modifying a storyboard or a cardboard mock-up requires no particular skill. Collaborating on physical prototypes not only increases participation in the design process but also improves communication among team members and increases the likelihood that the final design solution will be well accepted by both clients and users.

Precision

Prototypes serve as explicit representations of the design and help designers, engineers and users reason about the system being built. By their nature, prototypes require details. A verbal description such as "the user opens the file" or "the system

displays the results" provides no information about what the user actually does or sees. By contrast, prototypes force designers to *show* the interaction: just how does the user open the file and what are the specific results that appear on the screen?

Precision refers to the relevance of details with respect to the purpose of the prototype. It defines the tension between what the prototype states (relevant details) and what the prototype leaves open (irrelevant details). What the prototype includes is subject to evaluation; what the prototype leaves open encourages further discussion and design space exploration.

The form of the prototype must be adapted to the desired level of precision. The literature often uses the terms *low-fidelity* and *high-fidelity* (or lo-fi and hi-fi), but precision can involve intermediate levels as well. Some *mixed-fidelity* prototypes include elements of both. For example, when sketching the layout of an interface, the relative positions of the elements are relevant, but their specific labels are not. The hand-drawn prototype (Fig. 5:left) uses generic terms and squiggles instead of the final text. Here, the focus is on the overall layout and use of background color, not the final wording or style. By contrast, the FIGMA prototype (Fig. 5:right) is a high-fidelity prototype with all the specific details of the layout rendered in detail.

We distinguish among three levels of precision: *low-fidelity* prototypes that limit detail; *mixed-fidelity* prototypes that include detail only on selected interface elements; and *high-fidelity* prototypes that include all possible relevant detail.

- *Low-fidelity prototypes* are typically rough sketches or physical prototypes with very few details, intended to give a quick, overall impression of what a particular screen or aspect of the system might look like.
- *Mixed-fidelity prototypes* render certain aspects of the interface with rich detail and quickly sketch the rest. Some designers "sketchify" a computer-generated layout, to show which parts have been fixed and which remain open to further iteration.
- *High-fidelity prototypes* are usually created with a computer and include all the relevant visual details of what the screen or the hardware will look like.

The level of precision typically increases as successive prototypes are developed and more and more details are set, which means that the level of precision is often correlated with the type of representation. Thus, physical prototypes tend to be sketchy and imprecise, whereas software prototypes are more precise, and functional prototypes are usually very precise. Note, however, that this is not always the case. The designer can print a screen from a nonfunctional software prototype, such as from FIGMA, SKETCH or Adobe XD, and make it "interactive" by adding "buttons" and "menus" made of sticky notes. By contrast, a working functional prototype may ignore visual style, perhaps by including generic titles and using a monospace font, and focus only on how different features are implemented.

Hand-drawn physical prototypes can be drawn extremely precisely (Fig. 6:left), and three-dimensional physical prototypes can be approximate and imprecise (Fig. 6:right). Of course, physical prototypes are not a panacea. Sometimes, software

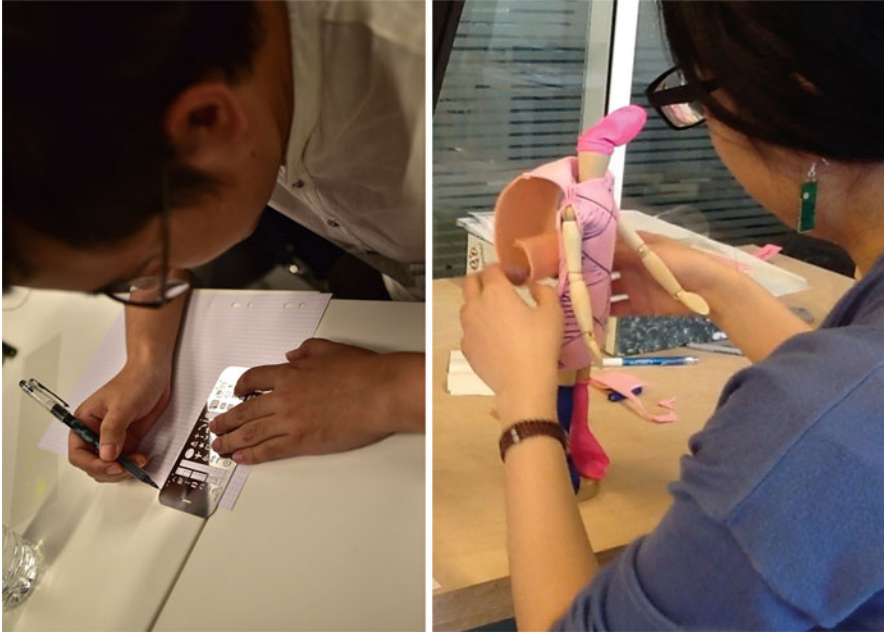

Fig. 6 Left: creating a hi-fi (high-precision) 2D physical prototype. Right: creating a lo-fi (low-precision) 3D physical prototype

or hardware prototypes are necessary to evaluate a particular design idea, especially when interactivity (Section "Interactivity") is complex. For example, systems that require a rapid response to user input or complex, dynamic visualizations may require more elaborate working prototypes. Software prototypes can also take advantage of animations or trigger transitions when the user presses or clicks on a button.

Interactivity

Designing effective interaction is difficult. While graphic designers can build on centuries of visual design, interactive software is still relatively new, and the interactive capabilities of technology are constantly evolving, especially with the advent of touch-based mobile devices, virtual and augmented reality, full-body interaction, etc. The quality of interaction is also tightly linked to users and requires a deep understanding of their work practices. For example, a video editor designed for a professional videographer requires a different interaction design than one designed for teenagers. Designers must take the context of use into account when designing the details of the interaction.

Designers need to consider how users will interact with their designs under different conditions of use. Prototypes that explicitly consider interactivity help designers avoid creating frustrating or difficult-to-use systems. Although non-interactive prototypes, such as wireframes, can illustrate how users will react to alternative design decisions, we recommend making prototypes *feel* interactive to the user. We distinguish among three types of interactivity: *noninteractive* prototypes that simply illustrate interaction possibilities, *fixed-path* prototypes that test pre-specified interaction paths; and *open* prototypes that allow users to experiment with a variety of interaction options.

- *Noninteractive prototypes* do not support interaction directly, but instead help designers consider different interaction possibilities, especially to illustrate or evaluate scenarios of future use. Designers often create *video prototypes* in the earliest design phases (Fig. 7:left), where participants can play the role of users who interact with physical prototypes (Fig. 7:right). In later design phases, designers create more refined, pre-computed animations to illustrate potential interaction with nonfunctional software prototypes.
- *Fixed-path prototypes* support limited interaction, following a pre-determined route through the interface. For example, the designer may create a storyboard that illustrates a series of successive states of the interface, either hand-drawn on paper or generated with a design tool (Fig. 8.) The user then “interacts” by touching the relevant part of the drawing or screen, at which point the designer presents the subsequent page or screen, perhaps with an animation. More sophisticated versions of this approach can provide multiple alternatives at each step, an effective strategy for testing future scenarios.

Fig. 7 Left: two designers manipulate the interface, while another shoots a video prototype. Right: one designer pretends to “resize” a paper window, while another designer shoots the interaction

Fig. 8 Storyboards illustrate a series of interactions with the system in the form of a scenario. Left: drawing a storyboard, with sticky notes that explain the interaction. Right: working out the details of the interaction by following the storyboard

- *Open prototypes* support a wide variety of interactions, usually based on a functional prototype that has implemented some, but not necessarily all of the features of the real system. Here, the user can interact with different parts of the system or compare different ways of accomplishing particular tasks. Although these prototypes may appear real, they are usually incomplete, with limited error-handling or reduced performance relative to the final system.

Designers can create prototypes with different levels of interactivity to address design challenges at different points of the design process. Noninteractive prototypes let designers focus on what the proposed interaction will look like and include different kinds of representation and levels of precision, ranging from quick videos of the interaction with rough physical prototypes in the earliest phase of the design to highly polished animations or marketing videos that present an idealized view of the future system at the end of the design process. Fixed-path prototypes are more common in the middle phase of the design process, since they provide designers and users with the experience of interacting with the prototype, but only in pre-specified situations. These are most useful when designers must choose among alternative forms of interacting with the system. Finally, open prototypes are only possible near the end of the design process, since they require extensive technical work to create them, but are extremely useful for discovering usability problems and understanding the details of how users will interact with the system in real-world settings.

Designers often create maps that consider how the user will transition from one state of the interface to the next. Figure 9 (left) shows a hand-drawn sketch of a series of wireframes that show how the user will move from screen to screen. Figure 9 (right) shows the same system, but rendered as a higher-precision, nonfunctional software prototype, using FIGMA. Clicking on one screen enlarges it and lets the user navigate to different screens.

One of the most important roles of a prototype is to illustrate how the user will interact with the future system. While it may seem easier and more natural to explore interaction with fully functional working prototypes, it is essential to work out

Fig. 9 Left: hand-drawn, low-fidelity sketch of a website. Right: high-fidelity nonfunctional prototype of the same website

the details of the interaction before committing to a major software development effort. Although designers can use any form of prototype to explore interaction, manipulating physical prototypes is by far the fastest.

The most common approach is the so-called the “Wizard of Oz” method (see Section “Wizard of Oz”), where one person, the “Wizard,” presents screens and manipulates the prototype in response to the actions of the person playing the role of the “user” (Fig. 10:left). This lets the design team explore different interaction possibilities and test whether certain actions are particularly easy or difficult.

However, as noted above, while physical prototypes can create highly sophisticated representations of the system, sometimes more highly developed software prototypes are necessary. Designers can also take advantage of the *Wizard of Oz* method with functional prototypes that implement certain interactions with the system (Fig. 7:right).

Life Cycle

Prototypes have different life spans – some are created quickly and thrown away when no longer needed, and others play an ongoing, evolving role throughout the design process. We distinguish among *rapid* prototypes that are created for a specific purpose and then discarded; *iterative* prototypes that evolve, either to work out some details or to explore various alternatives; and *evolutionary* prototypes that are designed to become part of the final system.

- *Rapid prototypes* are typically physical or nonfunctional software prototypes that are inexpensive and easy to produce. Typically created during the early and middle design phases, they allow designers to quickly explore a wide variety of possible alternatives, without investing too much time in creating them.

Rapid prototypes are often created to inform a particular design choice, such as ensuring that a specific interaction can be implemented with a sufficient level of performance.
- *Iterative prototypes* are developed as a reflection of the design as it progresses through each iteration. Designing such prototypes can be difficult given the tension between evolving toward the final solution and exploring an unexpected new design direction, any of which may be adopted or thrown away completely. Some prototype iterations thus explore different variations of the same theme, while others increase precision and work out the finer details of the interaction.
- *Evolutionary prototypes* are a special case of iterative prototypes, where the prototype evolves into part or all of the final system, such as *Agile* (Abrahamsson et al. 2017; Fowler and Highsmith 2001) programming projects that tightly couple design and implementation. These functional prototypes require more planning and practice than the above prototypes, since they act both as representations of the final system and the final system itself. In general, these prototypes should be combined with lighter weight prototypes, since evolutionary prototypes are relatively expensive and make it more difficult to explore alternative designs.

Scope

The scope of a prototype refers to the part of the interactive system that it covers. Designers must decide what role prototypes should play with respect to the final system, as well as the order of creating different prototype elements. We distinguish among *horizontal* prototypes that provide an overview of the entire system, *vertical* prototypes that focus on a particular feature or interaction and *path-based* prototypes that illustrate specific scenarios of the use of the system in a realistic context.

- *Horizontal prototypes* help designers represent the entire interactive system to get an overview of the design. They are particularly useful to get an overall picture of the system from the user's perspective and address issues such as consistency (similar functions are accessible through similar user actions), coverage (all required functions are supported) and redundancy (the same function is/is not accessible through different user actions). Early horizontal prototypes are usually physical and then refined into software prototypes. They can also be functional, e.g., by using an interface builder without creating the underlying functionality in order to get an early sense of the overall interface. Such functional prototypes tend to be evolutionary, as functionality is progressively added to transform them into the final system.
- *Vertical prototypes* help designers explore specific features or interactions in depth. The goal is to assess the feasibility of the feature or interaction, including testing it with real users. Vertical prototypes are generally high-fidelity, functional prototypes because their goal is to validate an idea at the system level.

They are usually created early in the project, before the overall architecture has been decided and focus on a single design question at a time.

- *Path-based prototypes* help designers explore specific scenarios of how the system would be used in a real-world setting. Scenarios are stories that describe a sequence of events and how the user reacts. A good scenario includes both common and unusual situations and should explore patterns of activity over time. Some scenarios also focus on potential breakdowns, to help designers consider problems users may encounter under different circumstances. Path-based prototypes focus on the features and interactions relevant to the scenario and range from noninteractive pre-recorded video prototypes (Fig. 10:left) to fully interactive, Wizard of Oz prototypes (Fig. 10:right). The goal is to create realistic situations where users can experience the system in real-world contexts, even if they only address a subset of the planned functionality, and are useful throughout the design process.

Summary

Table 1 summarizes the taxonomy of prototypes described above. The values of each dimension roughly match progress in the design process, from early stages where physical, low-fidelity, rapid prototypes help designers explore the design space and make early decisions to the final design stages where high-fidelity functional

Fig. 10 Wizard of Oz: Left: one designer manipulates a paper prototype for another designer, who plays the role of the user. Right: a design team tries a functional prototype, as they follow a scripted scenario

Table 1 A taxonomy of prototypes for interactive systems

Dimension	Type of prototype		
Representation	Physical	Nonfunctional	Functional
Precision	Low fidelity	Mixed fidelity	High fidelity
Interactivity	Noninteractive	Fixed-path	Open
Life cycle	Rapid	Iterative	Evolutionary
Scope	Horizontal	Vertical	Path-based

prototypes can evolve into the final product. Note, however, that design is highly iterative and it is perfectly possible, indeed often necessary, to create prototypes with different combinations of dimensions. So a designer can, for example, create a high-fidelity physical prototype that uses screens printed from a computer in a paper-prototyping session, and a functional prototype may be developed with the explicit goal of throwing it away, in order to test a design idea. Designers are encouraged to mix and match these dimensions to create the optimal type of prototype for a particular set of design questions at different points in the design process.

Methods and Tools for Rapid Prototyping

The goal of rapid prototyping is to explore ideas in a fraction of the time it would take to develop a working system. Shortening the prototype-evaluation cycle lets the design team evaluate more alternatives and iterate the design several times, which increases the likelihood of finding a solution that truly meets users' needs. Rapid prototypes also help cut off unpromising design directions, saving time and money. Rejecting an idea based on a rapid prototype is far easier than rejecting a more fully developed software prototype, functional or not. The methods and tools presented in this section are organized according to the three types of representation described in the previous section, from most rapid to most elaborate, including physical prototypes, as well as both nonfunctional and functional software (and hardware) prototypes.

Physical Rapid Prototyping Methods

Because they do not involve software, physical prototyping methods are often used as a tool for thinking through the design issues, to be thrown away when they are no longer needed. We describe four methods: simple paper and pencil sketches, three-dimensional foam or cardboard mock-ups, Wizard of Oz simulations and video prototypes.

Paper and pencil sketches are the fastest form of prototyping: designers simply sketch the interface using paper, transparencies and sticky notes (Mackay 2020) and use simple visual effects to animate them in response to the user's actions (Fig. 11). For example, a tiny arrow drawn at the end of a long strip cut from a transparency makes a handy mouse pointer that the designer can move in response to the user's actions. Sticky notes, with pre-prepared lists, can act as "pop-up menus." A rectangular cutout in a sheet of paper can be used to illustrate scrolling by sliding another sheet behind it with the content being scrolled. Playing the roles of both the user and system helps designers assess multiple layout and interaction alternatives in a very short period of time. For designers who lack visual design training, we recommend Greenberg et al. (2010), which offers a systematic approach for learning how to sketch interactive devices and systems.

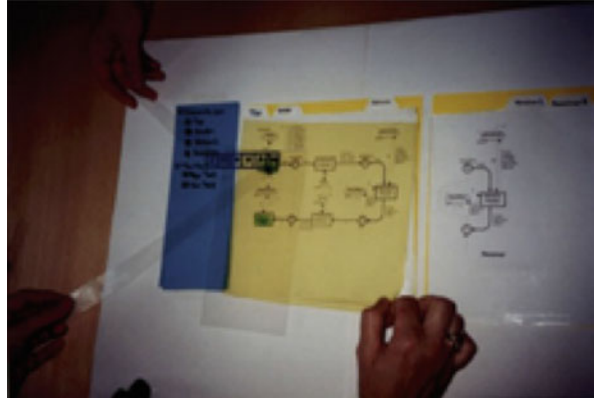

Fig. 11 Multiple designers manipulate transparencies to show what happens when the user moves the mouse, represented by an arrow drawn on another transparency

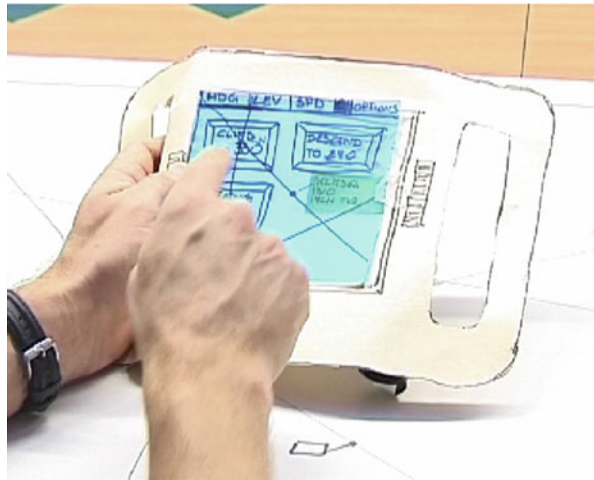

Fig. 12 A cardboard mock-up of an interactive screen from a participatory design project for air traffic controllers at Eurocontrol

Mock-Ups

Mock-ups are physical prototypes created with cardboard, foam-core or other found materials to simulate physical interfaces on mobile devices and other interactive objects. Creating and interacting with a mock-up helps designers explore how users will interact with the physical device and is especially helpful for simulating the use of mobile devices in different future settings. Mock-ups may be fully simulated, as in Fig. 12, or take advantage of existing devices, as in Fig. 4, which adds sticky notes and dots to a smartphone to ensure that the interface is effective at that small scale.

Wizard of Oz

The *Wizard of Oz* method (Kelley 1983) gives users the impression that they are working with a real system, even before it exists. The name comes from the scene in the 1939 movie of the same name where the heroine Dorothy encounters the

"Wizard of Oz," a giant green human head who breathes smoke and speaks with a deep voice. She later discovers that the Wizard is in fact a frail old man, the "man-behind-the-curtain" who pulls levers to make the mechanical Wizard of Oz move and speak.

The software version of the Wizard of Oz operates on the same principle. A user sits at a screen and interacts with what appears to be a working program. A designer at another computer plays the wizard, watching what the user does and responding directly to the user's actions. The wizard thus creates the illusion of a working software program. Alternatively, the wizard can operate from the same computer, for example by using a display that mirrors what the user sees, or the user can interact with a projected image operated by the wizard. This method lets users interact with partially functional computer systems in a realistic way. Note that designers may choose whether or not they reveal that the system is actually controlled by a person. However, for ethical reasons, it is important that the designer debriefs users appropriately after the session.

Video Prototyping

Video prototypes (Mackay 1988) use video to illustrate how users will interact with the new system in a real-world context. Video prototypes may build upon paper and pencil prototypes and cardboard mock-ups and can also use the Wizard of Oz method as well as images of real-world settings. Designers use video prototypes to explore the user's interaction with the system, including highlighting and assessing potential breakdowns and workarounds. We distinguish these from *tutorial* or *"how to"* videos that show how to perform specific tasks independent of context, and from *marketing* videos that show an idealized version of the future system without examining potential problems. Designers should treat video prototypes as an opportunity for thinking deeply about the proposed system from the user's perspective.

Creating a video prototype begins with making a *storyboard* that presents the scenario as a series of illustrations (either of the system or of users interacting with it), as well as accompanying dialog and instructions for shooting the video (Fig. 8). Designers shoot video prototypes using a variation of the Wizard of Oz method, where a camera person shoots video of one or more users as they interact with the physical or software prototype while other designers manipulate the prototype in response to the user(s) actions.

A storyboard, even an informal one, is essential for guiding the shoot. In our experience, a small design team with a well-designed storyboard can shoot a 3-5-minute video prototype in an hour or two, with no post hoc video editing. However, the same video prototype can take many hours or even days if shot without a storyboard, and post hoc editing is usually required. Title cards, as in silent movies, separate the clips or sections of the storyboard and make it easier to navigate the video. A typical video prototype starts with a title, followed by an establishing shot that shows the user in the context defined by the scenario. Next, a series of close-up and mid-range shots, interspersed with title cards, show the actual interactions.

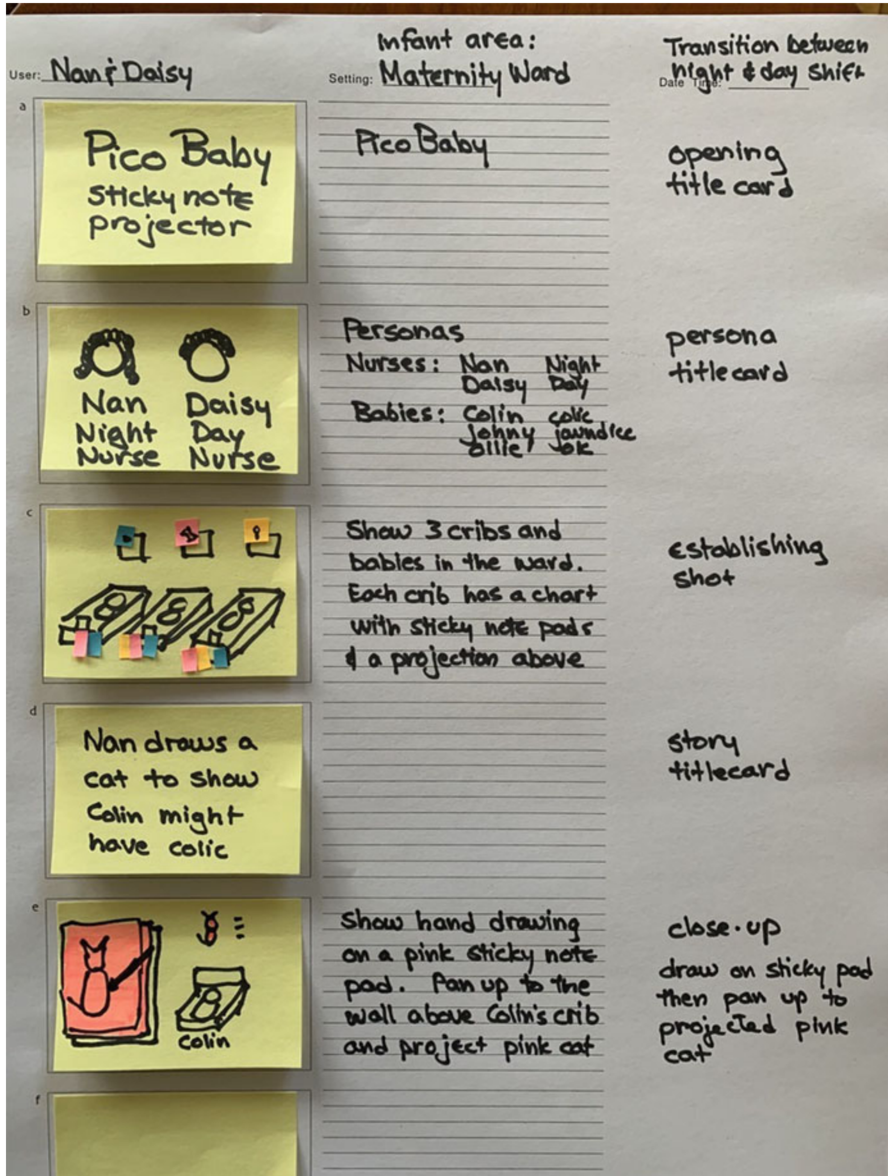

Fig. 13 A storyboard for a video prototype

A final title card at the end holds the credits. Figure 13 shows an example from Mackay (2020), which provides more detailed examples and guidelines for creating video-based prototypes.

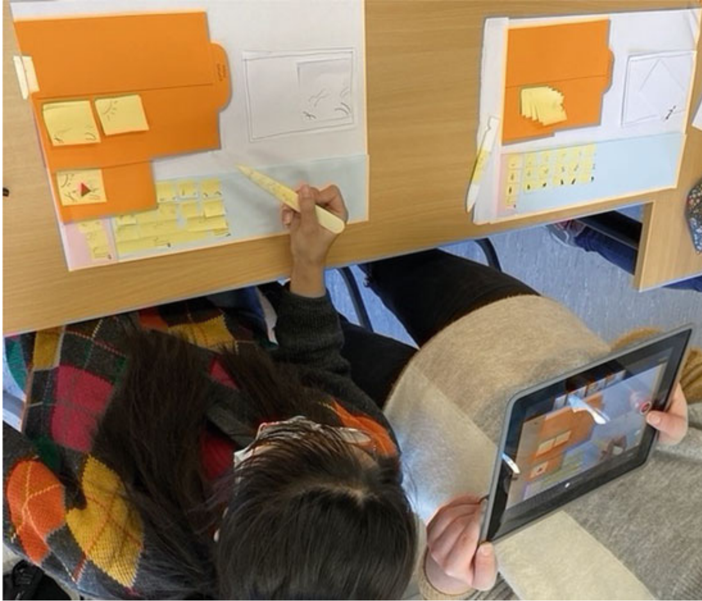

Fig. 14 One designer plays the role of the user, while the other shoots a video clip

Video prototypes take several forms. In some, a narrator explains each event, and several people on the sidelines may be necessary to move images and illustrate the interaction. In others, actors simply perform the movements and the viewer is expected to understand the interaction without a voice-over. Simple special effects can help illustrate the interactions. "Time-lapse photography" makes images appear and disappear based on the user's interaction. For example, to show that pressing a button pops up a window, first shoot a clip of the user pressing the button. Then, without the user moving their hand, add the pop-up window and proceed to shoot the next clip. When played back, the video will give the illusion of immediate system feedback. Figure 14 shows one designer shooting video of another designer who performs the role of the user. For higher-fidelity video prototypes, one can use more sophisticated techniques. For example, MONTAGE (Leiva and Beaudouin-Lafon 2018) uses chroma keying to combine video with digital animated sketches.

Combining Wizard of Oz and video is a particularly powerful prototyping method because it gives the person playing the user a real sense of what it might actually feel like to interact with the proposed tool, long before it has been implemented. Seeing a video clip of someone else interacting with a simulated tool is more effective than simply hearing about it; but interacting with it directly is more powerful still. Video prototyping is also effective for showing software developers how users will interact with the functionality envisioned by the designers and reproduce it as faithfully as possible. This is particularly useful when moving from physical to functional prototypes.

Rapid Nonfunctional Software Prototyping Methods

The goal of rapid nonfunctional software prototyping is to create higher-precision prototypes than can be achieved with physical prototyping methods. Such prototypes are important for communicating ideas to key stakeholders, not only users and developers but also clients and managers. They also help the design team fine-tune the details of each layout or interaction and highlight any design problems that were not apparent in less precise prototypes. Some nonfunctional software prototypes are low-fidelity prototypes that require highly dynamic interactions or visualizations that would be difficult or impossible to create with a paper-based physical prototype.

This section describes methods according to their level of interaction, beginning with noninteractive animations, followed by interactive simulations that provide fixed or multiple-path interaction and concluding with scripting languages that support open-ended interaction.

Animations

Animations represent what a person would see of the system if he or she were watching over the user's shoulder. They are useful when physical prototypes, including video, fail to capture a particular aspect of the interaction and it is important to have a quick prototype to evaluate the idea.

Animations can be created with specialized tools such as PRINCIPLE, but also with any authoring tool that can create images. For example, a slide-making application such as Microsoft POWERPOINT can be used to draw successive states of the interface in separate slides. Playing the slide deck then illustrates how the interface would behave. In a similar vein, many web designers use Adobe PHOTOSHOP to create simulations of their web sites. PHOTOSHOP images are composed of layers that overlap like transparencies. The visibility and relative position of each layer can be controlled independently. Designers can quickly add or delete visual elements, simply by changing the characteristics of the relevant layer. This permits quick comparisons of alternative designs and helps visualize multiple pages that share a common layout or banner.

Even a spreadsheet program can be used for prototyping: Berger (2006) describes the use of Microsoft EXCEL to prototype form-based interfaces, taking advantage of the table structure to create grid layouts. The designer can use EXCEL's workbook feature to create multiple pages accessible by tabs. Quickly flipping among the tabs illustrates the effect of different interactions.

Interactive Simulations

Interactive simulations are digital artifacts that support limited interaction and can be created without programming. Designers can use tools such as Adobe PHOTOSHOP and Microsoft POWERPOINT to create simple interactive simulations. With PHOTOSHOP, the effect of dragging an icon with the mouse can be obtained by placing the icon of a file in a layer and by moving the layer with the PHOTOSHOP

panning tool. With POWERPOINT, animations and transitions can be triggered by clicking on the slide. Multiple-path interactions can be created by attaching different animations to different areas on the slide.

Historically, richer tools such as Apple HYPERCARD (Goodman 1987) and Macromedia DIRECTOR were heavily used to create interactive simulations. HYPERCARD was based on a card metaphor. Cards shared a background and could contain input fields and buttons, with associated actions. Macromind DIRECTOR was based on a timeline where animations triggered by user actions could be attached to sprites. Both HYPERCARD and DIRECTOR featured a scripting language for programming advanced behaviors.

More recently, a new breed of tools designed specifically for creating interactive simulations has emerged. FIGMA, SKETCH, and Adobe XD are now widely used by designers of visual interfaces. They are based on the notion of views (or frames) that represent different states of the interface. Views contain visual and interactive elements and interactions such as clicks can trigger the switch to another view. These tools come with large collections of components that can be easily reused, such as interface elements that correspond to the visual aspect of the standard platforms. However, these tools support only a limited set of interactive behaviors, which leads to extremely standardized interfaces where most interactions are based on clicks and taps. As with development tools (see below), these tools tend to shape the designs, reducing rather than expanding the design spaces that can be explored.

Scripting Languages

Scripting languages are the most advanced online rapid prototyping tools. They make it easy to create throw-away prototypes in as little as a few hours or a few days. Scripting languages are complete programming languages that are both lightweight and easy to learn. They are interpreted, which means that changes to the program can be tested immediately. They have all the power of a full-fledged programming language, enabling the programming of advanced interactions. Widely available libraries add functionalities to the language, facilitating programming. For example, the D3.JS (Bostock et al. 2011) library for JAVASCRIPT is widely used to create interactive visualizations.

TCL and TK (Ousterhout 1994) provided an excellent language and user interface library for quickly creating graphical user interfaces. However, although they are still used by PYTHON, they are now considered quite dated. More recent and widely used scripting languages for interaction include PROCESSING (Reas and Fry 2010) and JAVASCRIPT. PROCESSING was created primarily for visual artists but supports simple forms of interaction and is particularly well-suited for exploring rich, real-time mappings between user input and visual output, for example, controlling a visualization with the x,y position of a mouse cursor.

JAVASCRIPT is “the language of the web.” It complements HTML and CSS, which describe web content and web layout, to define the interactive behavior of the content of a web page. A huge ecosystem has developed around JAVASCRIPT, with

hundreds of libraries. However, as with the design tools described in the previous section, most of these libraries target "standard" interactions based on clicking menus and buttons and filling out forms. JAVASCRIPT can nevertheless be used to create more creative and nonstandard interfaces, although this requires in-depth knowledge of HTML and CSS. Since many modern interactive applications are web applications that run in a web browser, prototypes developed with JAVASCRIPT often are evolutionary prototypes that are progressively transformed into the final application.

Tools for Developing Functional Prototypes

Prototypes may also be developed with traditional software development tools. In particular, high-fidelity prototypes usually require a level of performance that cannot be achieved with the rapid online prototyping methods described above. Evolutionary prototypes, which are intended to become the final product, require more traditional software development approaches. Finally, even shipped products are not "final," since subsequent releases can be viewed as initial designs for prototyping the next release.

The lowest-level tools are *graphical libraries* that provide hardware independence for painting pixels on a screen and handling user input, and *window systems* that provide an abstraction (the window) to structure the screen into several "virtual terminals." *User interface toolkits* structure an interface as a tree of interactive objects called widgets, while *user interface builders* provide an interactive application to create and edit those widget trees. *Application frameworks* build on toolkits and UI builders to facilitate the creation of typical functions such as cut/copy/paste, undo, help, and interfaces based on editing multiple documents in separate windows. *Model-based tools* semiautomatically derive an interface from a specification of the domain objects and functions to be supported. Finally, *user interface development environments*, or UIDEs, provide an integrated collection of tools for developing interactive software. This section covers the three tools most relevant to prototyping: user interface toolkits, user interface builders and application frameworks.

Note that it is not always best to use the highest-level available tool for prototyping. Higher-level tools constrain the types of interfaces that can be created. For example, user interface toolkits contain a limited set of "widgets" and application frameworks typically assume a stereotyped application with menus and a main window. High-level tools are most valuable in the long term because they make it easier to maintain the system, port it to various platforms or localize it to different languages. But these issues are irrelevant for vertical and throw-away prototypes, so a higher-level tool may prove less effective than a lower-level one.

Finally, developers must fully master these tools, especially when prototyping in support of a design team. Success depends upon the programmer's ability to quickly modify the details or the overall structure of the prototype. A programmer will always be more productive when using a familiar tool than if forced to use a more powerful but unknown tool.

User Interface Toolkits

User interface toolkits are libraries that implement the standard "look and feel" of the typical applications of a particular platform. All major platforms (Linux, macOS, Windows, iOS and Android) come with at least one standard UI toolkit. The main abstraction provided by a UI toolkit is the *widget*. A widget is a software object that has three facets that closely match the MVC (Model-View-Controller) model (Krasner and Pope 1988): the presentation, which defines the visual aspect of the widget (the *view*); the behavior, which defines how it reacts to user actions (the *controller*); and the application interface, which links these reactions to application functions (the *model*).

One limitation of widgets is that their behavior is limited to the widget itself. Interaction techniques that involve multiple widgets, such as drag-and-drop, cannot be supported by the widgets' behavior alone and require specific support in the UI toolkit. Some advanced interaction techniques, such as toolglasses or magic lenses (Bier et al. 1993), break the widget model both with respect to the presentation and the behavior and cannot be supported by traditional toolkits. UI toolkits are therefore useful only when prototyping "standard" interfaces.

However, specialized toolkits go beyond the widget model to address specific needs. For example, the advent of microcontrollers such as the Arduino or Raspberry Pi has prompted the development of a number of toolkits for creating physical interfaces. d.tools (Hartmann et al. 2006) is an advanced toolkit that features a rich authoring environment for creating information appliances (Fig. 15).

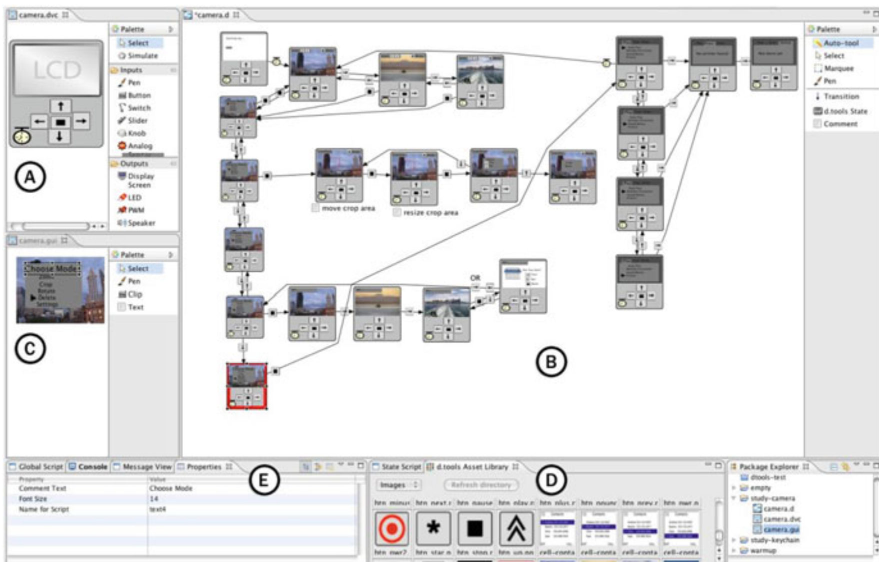

Fig. 15 The d.tools (Hartmann et al. 2006) authoring environment. A: device designer. B: storyboard editor. C: GUI editor. D: asset library. E: property sheet. (From Hartmann (2009), with permission)

User Interface Builders

User interface builders leverage user interface toolkits by allowing the developer of an interactive system to create the presentation of the user interface, i.e., the tree of widgets, interactively with a graphical editor. The editor typically features a palette of widgets that the designer uses to “draw” the interface as if it were a graphical editor. The presentation attributes of each widget and the overall layout can be edited interactively, as well as the mapping to application functions, if these are available. User interface builders save time that would otherwise be spent writing and fine-tuning rather dull code that creates widgets and specifies their attributes. It also makes it extremely easy to explore and test design alternatives.

Apple’s Interface Builder (Fig. 16) is one of the most advanced and powerful interface builders. A functional prototype can be created quickly by dragging widgets from a palette into the application window and by dragging connectors between these widgets and the application objects available in a separate palette. A significant part of the behavior of the interface can therefore be created interactively, without writing any code. The application can be tested at any time by switching the builder to test mode, making it easy to verify that it behaves as expected.

User interface builders are widely used to develop prototypes as well as final applications, which makes them well-suited to evolutionary prototypes. However, despite their name, they do not cover the whole user interface. Therefore, they still require a significant amount of programming, a good knowledge of the underlying toolkit, and an understanding of their limits, especially when prototyping novel visualization and interaction techniques.

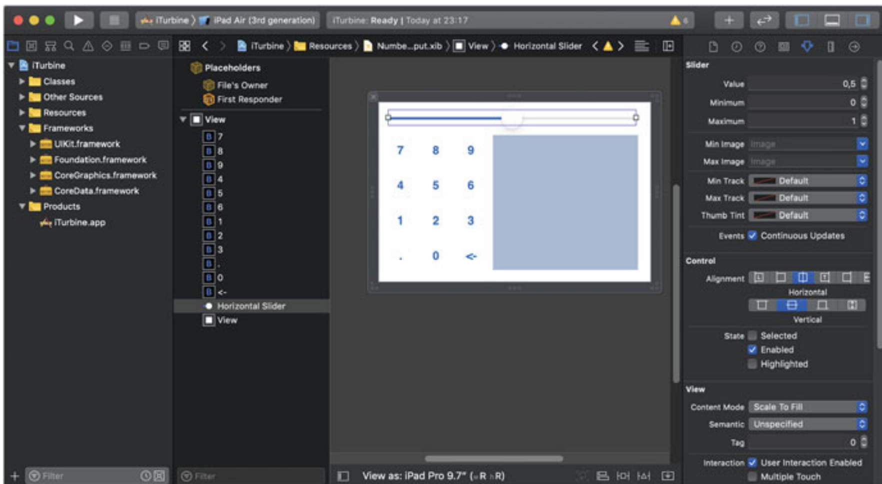

Fig. 16 Apple’s Interface Builder. The central pane contains the interface being built, the right pane contains an inspector to edit the selected widget, and the left panes contain the components of the project and of the user interface

Application Frameworks

Application frameworks are software libraries that provide a set of services and/or components to create an entire interactive application. They are usually based on a user interface toolkit, which they complement with support for features such as copy-paste, drag-and-drop, undo-redo and document management. They avoid the large amount of boilerplate code that is typically needed when using only a user interface toolkit. Developing a prototype with an application framework can be very rapid if the interface follows the patterns supported by the framework.

Popular frameworks for web-based applications include RAILS, REACT, ANGULAR, and VUEJS. A common feature of these frameworks is to simplify the binding between visual elements and the source code of the application so that, for example, the value typed in a text field is reflected in the value of a variable and vice versa (double binding). This facilitates the decoupling between the user interface elements and the rest of the application. For prototyping, this means that it is relatively easy to maintain a functional prototype as the design of the interface is iterated (Chatty et al. 2004). In some cases, it is even possible for a designer with limited programming knowledge to modify the interface, e.g., by editing the HTML and CSS parts of the prototype. As long as the visual elements are bound to the right variables and functions, the prototype will still be functional.

ENACT (Leiva et al. 2019) further facilitates the collaboration between designers and developers (Fig. 17). A designer can demonstrate a touch-based interaction and create a storyboard and a timeline with the key states of the interaction. The developer can then create a state machine and develop the necessary code to bring the interaction to life. The designer can then adjust parameters, test the interaction on the mobile device, and refine the design.

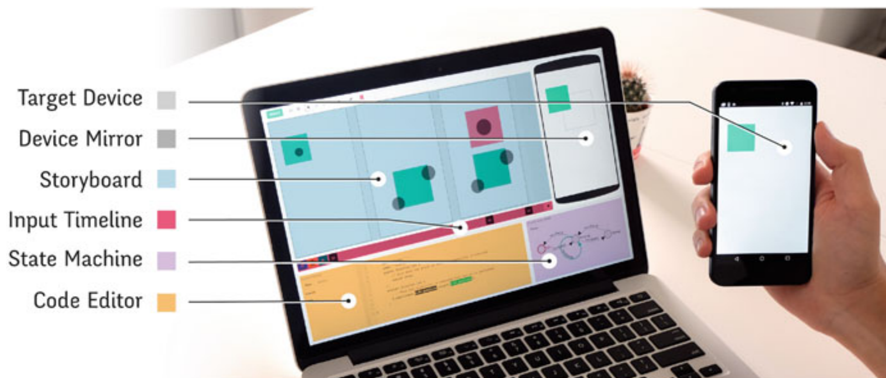

Fig. 17 ENACT (Leiva et al. 2019) uses a mobile device and a desktop interface with five areas: a storyboard with consecutive screens, an event timeline with a handle for each screen, a state machine, a code editor, and a device mirror

Summary

Table 2 summarizes the methods and tools described above according to the three *representations* of prototypes from the previous section. The *precision* and *interactivity* of the resulting prototypes increase for the first two rows from left to right. For the third row, since functional prototypes tend to be higher precision and more interactive than physical and nonfunctional prototypes, the tools are ordered from less to more powerful. It should be noted that scripting languages are becoming more and more powerful and efficient and are therefore now often used to create fully functional prototypes and even final applications. Some of these languages can be used by nonprogrammers, blurring the roles of designers and programmers in a design and development team. These teams are nevertheless encouraged to use the full gamut of methods and tools so as to combine the speed of rapid physical prototyping and the precision of functional prototypes according to the stage of the design process and the *scope* of the desired prototype.

Summary

Design is an active process of continually expanding and contracting the design space, where prototypes help designers envision new possibilities and then reflect upon and assess their design decisions. Prototypes are diverse and can fit within any part of the design process, from the earliest ideas to the final details of the design. Perhaps most important, prototypes offer an extremely effective means of communication between designers and other stakeholders. They serve as a fundamental component of participatory design, enabling users to participate and contribute at every stage.

This chapter describes the role of prototyping in participatory design, with the goal of actively involving users throughout the design process. We briefly discuss the benefits of collaborating with users to generate new ideas that expand the design space and selecting from among those ideas to contract the design space. We show how incorporating hands-on prototyping allows users from any age or background to contribute, both to clarify their needs in real-world contexts and to find realistic solutions.

Table 2 A taxonomy of methods and tools for rapid prototyping

Representation	Methods and tools		
Physical prototype	Mock-ups	Wizard of Oz	Video prototyping
Nonfunctional prototype	Animations	Simulation	Scripting language
Functional prototype	User interface toolkits	User interface builders	Application frameworks

We provide a taxonomy of prototypes and explain how to create and use them: *representation* refers to the physical form of the prototype, from a paper, cardboard or foam mock-up to a fully developed software or hardware simulation. *Precision* refers to the level of detail expressed in the prototype, from rough, hand-drawn sketches to highly polished images or objects, usually created with a computer. *Interactivity* refers to the level of interaction possible with the prototype, from simply showing what the interaction looks like without the ability to change it, to partially interactive scenarios where a user can follow a pre-defined path, to fully interactive Wizard of Oz simulations or operational prototypes. *Life cycle* refers to how the prototype evolves over time, from quick paper prototypes that are thrown away to more developed prototypes, some of which may become actual components of the final system. Finally, *scope* refers to which elements of the system are addressed by the prototype. This may include breadth-first horizontal prototypes, depth-first vertical prototypes or path-based story prototypes.

We conclude with a review of methods and tools for rapid prototyping, from making physical prototypes to creating video prototypes that express both the advantages and the potential breakdowns of the proposed system. We also describe diverse strategies for creating interactive prototypes with a variety of software tools.

In their chapter ▶ "User-Centred Design Approaches and Software Development Processes" in this volume, Lárusdóttir et al. categorize software development activities as analysis, design, implementation and testing. Although usually viewed primarily as a design activity, prototyping can also play an important role in all aspects of software development, albeit with a user-centered rather than a software-centered focus. For example, in the analysis phase, scenario-based video prototypes can express users' needs in realistic contexts. In addition to supporting all phases of design, from early, rough sketches to complete working functional prototypes, prototypes let developers translate the design as envisioned by the design team into working software. Prototypes created with interface builders or other interface design tools can often be incorporated directly into the final software, and prototypes of all forms can be used to test the system, and ensure that the system is, to paraphrase Buxton (2007), not only designed right, but in fact the right design.

The methods presented in this chapter have been used extensively in both research and industrial settings to create both novel interactive systems and real products. Our approach of combining participatory design with rapid prototyping has proven to be a very effective and low-cost way to develop innovative interfaces that truly match the users' needs.

Acknowledgments This work was partially supported by European Research Council (ERC) grants #321135 "CREATIV: Creating Co-Adaptive Human-Computer Partnerships" and #695464 "ONE: Unified Principles of Interaction" and is based on our earlier work on prototyping and participatory design (Beaudouin-Lafon and Mackay 2007).

References

Abrahamsson P, Salo O, Ronkainen J, Warsta J (2017) Agile software development methods: review and analysis. arXiv preprint arXiv:170908439

Bau O, Mackay WE (2008) Octopocus: a dynamic guide for learning gesture-based command sets. In: Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology (UIST'08). Association for Computing Machinery, New York, pp 37–46. https://doi.org/10.1145/1449715.1449724

Beaudouin-Lafon M, Mackay WE (2007) Prototyping tools and techniques. CRC Press, Boca Raton, pp 1017–1040

Berger N (2006) The excel story. Interactions 13(1):14–17. https://doi.org/10.1145/1109069.1109084

Bier EA, Stone MC, Pier K, Buxton W, DeRose TD (1993) Toolglass and magic lenses: The see-through interface. In: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'93). Association for Computing Machinery, New York, pp 73–80. https://doi.org/10.1145/166117.166126

Bødker S, Buur J (2002) The design collaboratorium: a place for usability design. ACM Trans Comput-Human Interact 9(2):152–169. https://doi.org/10.1145/513665.513670

Bostock M, Ogievetsky V, Heer J (2011) D^3 data-driven documents. IEEE Trans Vis Comput Graph 17(12):2301–2309. https://doi.org/10.1109/TVCG.2011.185

Buxton B (2007) Sketching user experiences: getting the design right and the right design. Morgan Kaufmann, San Francisco. https://doi.org/10.1016/B978-0-12-374037-3.X5043-3

Chatty S, Sire S, Vinot JL, Lecoanet P, Lemort A, Mertz C (2004) Revisiting visual interface programming: creating GUI tools for designers and programmers. In: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology, UIST'04. Association for Computing Machinery, New York, pp 267–276. https://doi.org/10.1145/1029632.1029678

Design Council (2005) A study of the design process – the double diamond. Technical report, Design Council UK. http://www.designcouncil.org.uk/sites/default/files/asset/document/ElevenLessons_Design_Council%20%282%29.pdf

Fowler M, Highsmith J (2001) The agile manifesto. Softw Dev 9(8):28–35

Goodman D (1987) The complete hypercard handbook. Bantam Books, New York

Greenbaum JM, Kyng M (1991) Design at work: cooperative design of computer systems. L. Erlbaum Associates Inc.

Greenberg S, Carpendale S, Marquardt N, Buxton B (2010) Sketching user experiences: the workbook. Morgan Kaufmann, San Francisco. https://doi.org/10.1016/C2009-0-61147-8

Hartmann B (2009) Gaining design insight through interaction prototyping tools. PhD thesis, Stanford University

Hartmann B, Klemmer SR, Bernstein M, Abdulla L, Burr B, Robinson-Mosher A, Gee J (2006) Reflective physical prototyping through integrated design, test, and analysis. In: Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST'06). Association for Computing Machinery, New York, pp 299–308. https://doi.org/10.1145/1166253.1166300

Hutchinson H, Mackay W, Westerlund B, Bederson BB, Druin A, Plaisant C, Beaudouin-Lafon M, Conversy S, Evans H, Hansen H, Roussel N, Eiderbäck B (2003) Technology probes: inspiring design for and with families. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'03). ACM, New York, pp 17–24. https://doi.org/10.1145/642611.642616

Kelley JF (1983) An empirical methodology for writing user-friendly natural language computer applications. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'83). Association for Computing Machinery, New York, pp 193–196. https://doi.org/10.1145/800045.801609

Krasner E, Pope S (1988) A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. J Object-Oriented Program 1(3):26–49

Laseau P (1980) Graphic thinking for architects and designers. Van Nostrand Reinhold
Leiva G, Beaudouin-Lafon M (2018) Montage: a video prototyping system to reduce re-shooting and increase re-usability. In: Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology (UIST'18). Association for Computing Machinery, New York, pp 675–682. https://doi.org/10.1145/3242587.3242613
Leiva G, Maudet N, Mackay W, Beaudouin-Lafon M (2019) Enact: reducing designer–developer breakdowns when prototyping custom interactions. ACM Trans Comput-Human Interact 26(3). https://doi.org/10.1145/3310276
Lim YK, Stolterman E, Tenenberg J (2008) The anatomy of prototypes: prototypes as filters, prototypes as manifestations of design ideas. ACM Trans Comput-Human Interact 15(2). https://doi.org/10.1145/1375761.1375762
Mackay W (1988) Video prototyping: a technique for developing hypermedia systems. In: Proceedings of CHI'88, Conference on Human Factors in Computing
Mackay W (2020) Designing with sticky notes. In: Christensen BT, Halskov K, Klokmose CN (eds) Sticky creativity: post-it® note cognition, computers, and design. Explorations in creativity research, science direct. Elsevier B.V, pp 231–256. https://doi.org/10.1016/C2017-0-04695-5
Muller M, Druin A (2002) Participatory design: the third space in HCI. In: The human-computer handbook: fundamentals, evolving technologies and emerging applications. Julie A. Jacko and Andrew Sears, editors. Lawrence Erlbaum Associates, Inc, Hillsdale, pp 1051–1068
Muller MJ, Kuhn S (1993) Participatory design. Commun ACM 36(6):24–28. https://doi.org/10.1145/153571.255960
Norman D, Draper S (eds) (1986) User centered system design. Lawrence Erlbaum Associates, Hillsdale
Ousterhout J (1994) Tcl and the Tk Toolkit. Addison Wesley, Reading
Reas C, Fry B (2010) Getting started with processing. Make: community
Schuler D, Namioka A (1993) Participatory design: principles and practices. Lawrence Erlbaum Associates, Inc, Hillsdale