CHAPTER 2 BACKGROUND: PROTOTYPES IN THE DESIGN PROCESS

This dissertation proposes novel tools for the prototyping of user interfaces as part of a larger user interface design process. Doing so successfully requires understanding underlying principles and practices of design. This chapter presents a brief review of different models of design and the role prototypes play in the process.

2.1 DESIGN, DEFINED

User interface design is informed and influenced by professional design disciplines such as product design on one side and by software engineering on the other side. This section provides a brief overview of the history of professional design and introduces some established models of the design process to motivate the development of design-specific tools.

2.1.1 WHAT DO WE MEAN BY DESIGN?

Herbert Simon provided a very broad definition of design as "devising courses of action aimed at changing current situations into preferred ones" [230]. Countless competing definitions exist. Common to many definitions is the focus on a specific *process*, with the goal of *creating plans or models* for the creation of new artifacts, which have to *fit* potentially conflicting sets of constraints, requirements, and preferences. To elaborate on these three core characteristics:

1) Design is a process and has structure — there is a set of core activities designers engage in, regardless of the domain of design.
2) Design is not manufacturing — for physical artifacts, the final realization is done by someone else. For software, the division between design and implementation may be less clear. In both domains, the end product of design is often a specification that will be interpreted and implemented by someone else.
3) Design has a client and users — it is accountable to external judgment. Different stakeholders may have conflicting expectations.

Design is thus distinguishable as a unique discipline from art (creation which is accountable to the vision of the artist); engineering ("the application of scientific and mathematical

principles to practical ends" [31]); and science (the development of generalizable knowledge through observation, experimentation and hypothesis testing).

A more pragmatic characterization would be that *design is what professional designers do*. The field of design research adopts this perspective and describes the practices of successful practitioners to analyze what makes these practices effective. Cross [66], a prominent design researcher, argues that design has a "unique way of knowing" and distills four core abilities exercised by professional practitioners:

1) resolving ill-defined problems
2) adopting solution-focused cognitive strategies
3) employing abductive or appositional thinking
4) using non-verbal modeling media

In *ill-defined* or "wicked" [213] problems, the problem formulation itself is not clear at the outset and remains to be defined. Because the problem statement itself is not fixed, it is not possible to enumerate all possible options or to find an optimal solution. Simon argued that design problems therefore cannot be solved by optimizing, they can only be *satisficed* [230] — one can tell an adequate solution from an inadequate one, and make relative judgments of fit, but no global optimum exists.

Designers adopt *solution-focused strategies* by generating possible solutions first, then checking to what extent the generated ideas are adequate for the problem. Cross, reporting on a study of designers, summarizes: "Instead of generating abstract relationships and attributes, then deriving the appropriate object to be considered, the [designers] always generated a design element and then determined its qualities." [66:100]. Creation comes before analysis, and only through the creation of prototypes and other representations is it possible to test to what extent a design idea fulfills the design goals.

This tendency to produce proposals first is an expression of a*bductive reasoning* which, in contrast to deductive or inductive thinking, starts with concrete observations and guesses, which only later lead to theories about a design space. Making the right guesses or creative leaps requires experience.

Finally, designers tend think with, and communicate through *artifacts* and *models* rather than written language – sketches, diagrams, models and prototypes are used both to work through problems as well as to anchor communication with design team members and other stakeholders [66:28].

2.1.2 A SHORT HISTORY OF PROFESSIONAL DESIGN

Architecture has the claim to being the oldest design discipline. Its focus is on the holistic creation of structures that simultaneously satisfy requirements of functionality, economy, and aesthetics. Notably the architect is not the one who creates the building itself: her role is to transform needs, requirements, and constraints into a suitable plan that can then executed by a builder. Professional *product design* as a discipline emerged as a result of the shift from one-off artifacts created by craftspersons to mass production after the industrial revolution. While craftsmen would iterate from project to project and slowly evolve a product over time, mass production yielded many identical copies [175,237]. Because making changes to the tooling for mass manufacturing became more expensive, while marginal cost of production decreased, more care and planning was needed before manufacturing commenced to ensure that the manufactured product was in fact functional and desirable to consumers. Notable pioneers of product design in the first half of the 20th century include Henry Dreyfuss, Raymond Loewy, Walter Dorwin Teague, and Norman Bel Geddes. Their autobiographies offer detailed accounts of the mid-century industrial design process in North America [73,175,237].

Product design as a methodology has since been assimilated by the software industry. One of the formative academic works that advocated for this transfer of process was Winograd's "Bringing Design to Software" [251]. As software is ultimately used by people, its user interfaces should be created with the same concern for utility, usability, and satisfaction as other artifacts of daily life.

2.1.3 HOW DO DESIGNERS WORK? MODELS OF THE DESIGN PROCESS

How do the underlying principles of designerly knowledge introduced in section 2.1.1 find expression in designers' work practices? The process that evolved from architecture and product design is characterized by four core strategies: *need finding* through user research methods to establish constraints, *ideation* to generate many possible ideas and subsequently select promising ideas, *prototyping* to create concrete models and approximations based on those ideas, and *iterative refinement* based on testing generated prototypes. A more detailed model of this iterative process, as described by Moggridge [189], is shown in Figure 2.1.

Need finding involves learning about the target users of a new product — what are their unmet needs and unresolved pains; what are their motivations? Needs, requirements, and constraints may be expressed in narrative form, *e.g.*, as personas and scenarios [64:p. 123]; or more formally, *e.g.*, as user and task models [99]. The data gathered from such user research is

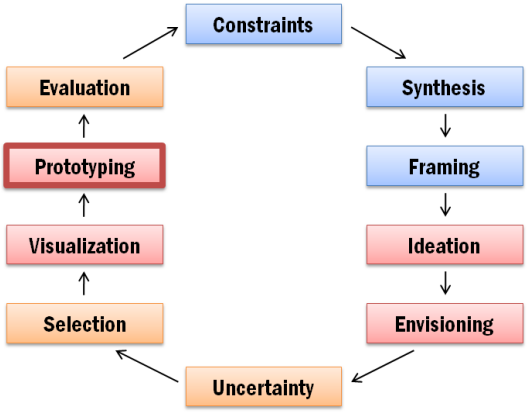

Figure 2.1: Design process stages according to Moggridge [189]. Diagram redrawn by the author.

then used to construct a concrete *point of view* or *framing* that encapsulates the goals a new design seeks to achieve. Given a framing, designers generate a multitude of concrete proposals. Initially, idea generation can take the shape of brainstorming or sketching of alternatives. To move from graphical envisioning towards concrete, testable artifacts, designers next generate concrete prototypes. These proposals are then compared an evaluated — against each other, or against user or stakeholder feedback. The gained knowledge is then used to drive the next iteration.

The process model described above is commonly observed, but by no means canonical. A wide-ranging overview of different design methodologies can be found in Jones [135]. Jones categorizes these methods and distills an important common thread: design is a sequence of divergent steps, where ideas are produced; and convergent steps, where ideas are eliminated. Buxton echoes this theme, writing that design alternates between concept generation and concept selection [55].

2.2 UNDERSTANDING PROTOTYPES

The prior section established that prototyping is a core activity in design across different domains. This section reviews some conceptions of prototypes in design and computer science and summarizes the literature on the purpose, role, and place of prototypes.

2.2.1 PROTOTYPES, DEFINED

The notion of a prototype is overloaded and there is no generally agreed upon definition. As a broad, inclusive definition, Moggridge regards a prototype as "a representation of a design, made before the final solution exists." [189]. Houde and Hill similarly point to the purpose of prototypes as indicators of a future reality, and distinguish between two functions — exploration and demonstration [126]. Buchenau and Suri add a third function of prototypes as tools for gaining empathy: "[A]n Experience Prototype is any kind of representation, in any medium, that is designed to understand, explore or communicate what it might be like to engage with the product, space or system we are designing" [52]. Lim and Stolterman foreground the role of prototypes as learning vehicles: "Prototypes are the means by which designers organically and evolutionarily learn, discover, generate, and refine designs." [170]

The above definitions place very little restrictions on the medium of the prototype or the attributes of a design it tries to represent. In the software engineering literature, prototypes are often defined more narrowly as working models, created in the same software medium as the final deliverable. Lichter writes that "Prototyping involves producing early working versions ('prototypes') of the future application system and experimenting with them" [166]. Connell and Schafer explicitly distinguish software prototypes from — in their view insufficient — other modeling media: "A software prototype is a dynamic visual model providing a communication tool for customer and developer that is far more effective than either narrative prose or static visual models for portraying functionality." (quoted in [208])

In contrast to the software engineering focus on producing functional software, Rettig [211] and Wong [255] advocate that user interface prototypes should not be constructed in software in early project stages. Both argue for low-fidelity paper-based prototypes. To better understand this multitude of viewpoints, this section summarizes prior publications about prototypes and prototyping in design in general, and within HCI and software engineering in particular.

2.2.2 BENEFITS OF PROTOTYPING

Are there concrete, measurable, defensible benefits of using a prototyping-driven design approach, as opposed to a more linear approach, *e.g.*, the waterfall model? This section reviews experimental and theoretical arguments for the benefit of prototyping.

2.2.2.1 Quantifying the Value of Prototyping

The ideal experimental result in favor of prototyping would be that prototyping leads to better design outcomes. However, operationalizing design quality in experimental settings is difficult and isolating the impact of prototyping has proven to be problematic for real-world design tasks. The most concrete result to date is reported by Dow et al. [72] who found that for a constrained experimental design task with a time limit and a concretely measurable outcome, participants who built early prototypes and iterated outperformed those who did not prototype. Dow's experimental task was the mechanical engineering egg-drop exercise — participants are asked to create a vessel that protects a raw egg from a vertical fall and subsequent impact, using a limited set of everyday materials. In a between-subjects design, the treatment group, which had to build a testable prototype early and was forced to iterate on that prototype, outperformed the control group, in which prototyping was not encouraged. In particular, novices unfamiliar with the task who prototyped performed as well as experts who did not prototype.

In the absence of other strong experimental results, a frequently cited benefit by proponents is that prototyping leads to earlier identification of problems and blind alleys, when it is still feasible to fix them. McConnell summarizes several studies that have shown that for software defects, the cost of finding an error increases by an order of magnitude for each product phase [183:29], and it appears reasonable to extrapolate similar costs to usability and user experience problems.

2.2.2.2 Cognitive Benefits of Prototyping

Research in Cognitive Science suggests that the construction of concrete artifacts — prototyping — can be an important cognitive strategy to successfully reason about a design problem and its solution space. This section presents some arguments for the cognitive benefits of prototyping.

ARGUMENT 1: WE KNOW MORE THAN WE CAN TELL.

Embodied cognition theory argues that thought (mind) and action (body) are deeply integrated and co-produce learning and reasoning [59,60,61]. In this view, "thinking through doing" — engaging with ideas on a tangible level — is a more successful strategy for design than thinking hard about the problem alone. Why might this be the case?

Polanyi argues that much of our expertise and skill are "action-centered" and as such not available to explicit, symbolic cognition. Polanyi introduced the term *tacit knowledge* to

describe such expertise. A well-known example is the problem of describing to someone else how to ride a bicycle. Riding a bike is an action-centered skill, one gained through repeated practice and one only accessible as an action in the context of sitting on a bike. Practicing designers such as Moggridge [189] argue that much knowledge in design is tacit and that designers therefore need to create concrete artifacts to express their tacit knowledge [207].

ARGUMENT 2: COGNITIVE ACTIVITY EXTENDS INTO OUR ENVIRONMENT.

Proponents of *distributed cognition* argue that what is cognitive extends beyond the individual and encompasses the environment, artifacts and other people [123,130]. Hutchins describes in detailed case studies how people solve hard problems by offloading tasks into appropriate artifacts in their environment. For example, medieval navigation was aided by the Astrolabe; airline navigation is a task distributed between pilot, co-pilot, and instruments.

In this view, designers need concrete artifacts such as prototypes to be more effective in their reasoning. Along the same lines, Hutchins also argues that "material anchors" help stabilize conceptual knowledge [131]: "Reasoning processes require stable representations of constraints. [...][T]he association of conceptual structure with material structure can stabilize conceptual representation."

ARGUMENT 3: ACTIONS IN THE WORLD CAN OUTPERFORM MENTAL OPERATIONS

Kirsh and Maglio introduced a distinction between pragmatic and epistemic actions [143]: pragmatic actions are those that advance us toward a known goal; epistemic actions in contrast uncover more information about the goal. Kirsh and Maglio showed, through a study of Tetris players, that external actions in the world can be faster or more efficient than mental operations. Their study measured the amount of piece rotations performed by novice and expert Tetris players, and found that experts rotated their pieces more frequently. Why? Because the cost of performing the rotation in the game and then visually comparing the shape of the piece with the shape of open gaps on the board was faster than mentally rotating and checking for fit. Similar results have been found for the game of Scrabble, where expert players rearrange their set of letters to help them reason about possible words that can be formed with that set. Constructing concrete prototypes could thus be faster than trying to reason about a design problem in the abstract.

2.2.2.3 Reflective Practice: The Value of Surprise

Schoen introduced the concept of *reflective practice* to describe designers' activity during visualization and prototyping [221]. Reflective practice is the repeated framing and evaluation

of a design challenge by working it through, rather than thinking it through. For Schoen, successful product and architectural designs result from a series of "conversations with materials." Here, the "conversations" are interactions between the designer and the design medium — sketching on paper, shaping clay, building with foam core. The production of concrete prototypes provides the crucial element of surprise, unexpected realizations that the designer could not have arrived at without producing a concrete manifestation of her ideas. Schoen terms this element of surprise "backtalk". The backtalk that artifacts provide helps uncover problems or generate suggestions for new designs.

2.2.2.4 Prototyping as a Teaching Technique

Prototyping has also been considered teaching technique that seeks to instill better design intuitions over time [136]. By continually forcing designers to be faced with the consequences of their actions through prototype testing, they are held accountable for their ideas. Designers thus develop a better sense for which ideas work and which do not.

2.2.3 THE PURPOSE OF PROTOTYPING — DESIGN PERSPECTIVES

What questions do prototypes answer? When and how should they be constructed? This section summarizes arguments from product design and human-computer interaction research. The subsequent section will present contrasting arguments from software engineering.

2.2.3.1 What Do Prototypes Prototype?

Houde and Hill [126] classified ways in which prototypes can be valuable to designers. Prototypes in their view include "any representation of a design idea, regardless of medium." Their model defines three types of questions a prototype can address: the *role* of a product in the larger use context; its *look and feel*; and its technical *implementation*. These questions are set up as end points in a triangular, barycentric coordinate design space into which prototypes are plotted (Figure 2.2).

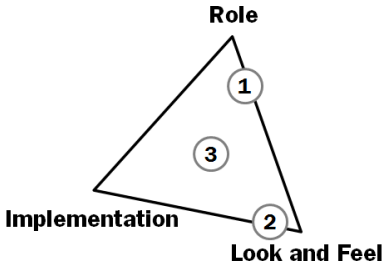

Figure 2.2: The Houde & Hill model distinguishes *Role*, *Implementation*, and *Look and Feel* functions of prototypes.

Role refers to questions about the function that an artifact serves in a user's life—the way in which it is useful to them. *Look and feel* is concerned with questions about the concrete sensory experience of using an artifact—what the user looks at, feels and hears while using it. Implementation refers to algorithms and engineering techniques used to realize functionality of a product — "the 'nuts and bolts' of how it actually works."

For reasons of economy, any given prototype will only address some of these aspects, or prioritize some over others. For example, a video clip that shows a "commercial" of an envisioned product in use would prioritize its role (Figure 2.2–1); a screen mockup of a new graphics application showing menus and toolboxes would prioritize look and feel (Figure 2.2–2); while a demonstration of algorithms required for that graphics application would prioritize implementation. Prototypes that strive to strike a balance and address all three questions are labeled "integration prototypes" (Figure 2.2–3). Such prototypes most closely approximate the final design and permit testing of the overall user experience, but are also most resource intensive to construct.

2.2.3.2 Experience Prototyping

Buchenau and Suri [52] introduced the term "Experience Prototyping" to refer to prototyping activity that enables stakeholders to gain first-hand experiential understanding of either design problems or of proposed solutions. An example of such a prototype given by the authors is wearing gloves while operating a consumer electronics device to experience the reduced dexterity of older adults. Experience prototypes focus on direct active bodily involvement of the designer or client in a constructed situation. Three uses for experience prototyping are described: understanding existing use; exploring future situations; and communicating designs to others.

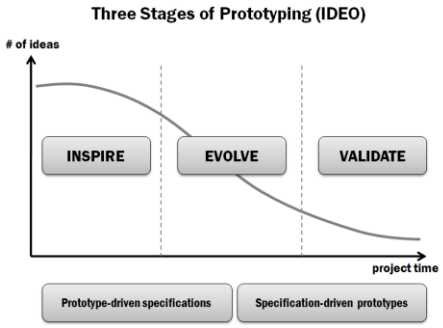

Figure 2.3: The IDEO three-stage model of prototyping: as a design project progresses, the number of entertained ideas decreases, and prototypes turn from inspiration tools to validation tools. Diagram redrawn by the author.

To understand existing situations that call for better design solutions, experience prototyping may involve role playing to gain empathy for target users. As an example, the authors cite the redesign of a remote control interface for an underwater camera vehicle. The existing experience was prototyped by one designer "playing" the vehicle with a shoulder mounted camera, and another designer yelling commands ("move up") and watching the video feed on a television monitor.

To explore future situations, Buchenau and Suri advocate creating multiple concrete artifacts or repurposing found artifacts and everyday objects. Designers on the project team have enough shared context to interpret these objects as stand-ins for future artifacts. For example, a pebble might be used to suggest a handheld wireless controller. However, if exploration requires input from external users, experience prototypes may have to be more specific and functional, as end users don't share the same background or conceptual framework with the team.

When communicating design solutions to clients and other external parties through prototypes, the intent is frequently to persuade. Such prototypes are often polished and complete and can take on the role of a "living specification." The authors caution that prototypes that succeed in conveying a complete experience can easily be mistaken to be a complete product.

2.2.3.3 Inspiration, Evolution, Validation

In personal communication, Hans-Christoph Haenlein, Director of Prototyping at IDEO, the prominent Bay Area design consultancy, described a company-internal three stage view of prototyping (Figure 2.3) [100]. In the beginning of a project, many parallel prototypes are

generated to get inspiration. Here, prototypes are often very dissimilar from each other to explore fundamentally different design options. Later on, a smaller number of ideas are iteratively evolved to resolve more focused design questions. Through both phases, project specifications are derived from the prototypes. Towards the end of a project, very complete prototypes are built to validate the design specification as a whole. Haenlein also makes an explicit distinction between prototypes used internally by the design team for exploration, and prototypes created for communicating design insights to external clients and other stakeholders.

Buxton [55] draws a distinction between *sketches* and *prototypes*. For him, sketches are "quick, timely, inexpensive, disposable, plentiful"; "they suggest and explore rather than confirm". Prototypes in contrast are "didactic, they describe refine, answer, test, resolve; they are specific and are depictions" [55:140]. While the distinction in nomenclature is unique to Buxton, the expressed difference between prototypes used for inspiration and those used for experimentation, evolution and validation matches the IDEO model.

2.2.3.4 Prototyping as Inquiry

Gedenryd stresses that prototypes are "inquiring materials", that is, materials with a cognitive purpose [84]. Many prototyping approaches all share the underlying goal to envision the future situation of the designed artifact in use — prototyping is thus a "situating strategy". Echoing distinctions drawn by Haenlein and Buxton, Gedenryd distinguishes between exploratory prototypes used to familiarize oneself with the problem, and experimental prototypes, which probe and test specific design hypotheses. He further distinguishes between horizontal relevance (breadth) and vertical relevance (depth) of the functionality explored in a prototype.

As a guideline, Gedenryd advocates that prototypes exhibit a minimalist approach: "A good prototype serves its purpose as a basis of inquiry and interactive cognition, while being simple to create. This means that it should have the properties required for its purpose, and as few other properties as possible. It also means that relevance is always relative to just what exactly a prototype will be used for; this determines what properties it will need to have." [84:165]

2.2.3.5 Low-Fidelity Prototypes Might Be Preferable

Rettig [211] and Wong [255] argue that the resolution or fidelity of a user interface prototype should match the level of detail of the questions asked of the prototype. In particular, Rettig

advocates against building functional software prototypes of user interfaces early on because their surface finish is too high at a time when the general resolution of the project is still low. According to Rettig, building functional UI prototypes ("high-fidelity prototypes") early on squanders design resources and yields the wrong kind of feedback. Particularly, Rettig cites four problems:

1) High-fidelity prototypes take too long to construct and modify.
2) Testers of the prototype are lead to comment on surface attributes such as typography and alignment, when those are not the attributes tested.
3) The act of constructing a high-fidelity prototype creates emotional investment by developers in that prototype, which results in resistance to act on feedback that asks for fundamental changes. Similarly, a high-fidelity prototype creates expectations by users exposed to the prototype that may be hard to change later.
4) High-fidelity prototypes are too brittle and have no graceful "repair strategies" if users run into bugs.

As an alternative, Rettig proposes paper prototyping of user interfaces, where interfaces are assembled out of different layers of cut out paper strips. A designer simulates the logic of the application by rearranging paper strips. Wong is also concerned with the fidelity of UI prototypes and suggests taking inspiration from graphic design by creating "rough" UI prototypes through sketching and omission of concrete details.

One fundamental shortcoming of paper-based UI prototyping is that the human "computer" who rearranges UI elements fundamentally changes the experience of interface dynamics. While useful for exploring questions of interface layout, content, and structure, paper prototypes are therefore less useful for exploring interactive behaviors in user interfaces.

2.2.4 THE PURPOSE OF PROTOTYPING — SOFTWARE ENGINEERING PERSPECTIVES

This section summarizes publications on prototyping from outside the field of human-computer interaction and product design. Not surprisingly, software engineering prototypes are more frequently concerned with testing implementation strategies than user experience. However, the software engineering literature also departs from human-computer interaction publications on prototyping in additional ways: prototypes are frequently seen as early version of the final software, rather than standalone artifacts to be discarded after testing. In

addition, more emphasis is placed on capturing and documenting what questions a prototype explored, and what was learned from it.

2.2.4.1 Exploration, Experimentation, Evolution

Floyd [79], in an early workshop on prototyping for complex software systems, describes two primary goals of prototypes: 1) functioning as "learning vehicles" and 2) enhancing communication between developers and users, as developer introspection of user needs often leads to inadequate products.

For Floyd, a software prototype must be functional enough to be demonstrated to users with "authentic, non-trivial tasks." That functionality may either be implemented, or simulated. In either case, Floyd assumes that for complex software projects, resource constraints only permit one such prototype to be built and tested at a time. Floyd also claims that by demonstrating a prototype to users, their expectations of the final system are "deeply influenced" so that the designer is committed to the overall outline of the prototype. This places the designer in a paradoxical situation: prototypes are constructed to learn, but their very construction constrains the extent to act on what was learned by modifying the design. This paradox may have been an artifact of the types of applications considered — custom software written for individual clients, so that the prototype testers and final users are identical.

Three different purposes of prototyping are distinguished by Floyd (Table 2.1): *exploration* (clarifying requirements, discussing alternatives), *experimentation* (measuring how adequate a proposed solution is), and *evolution* (adapting an existing system to changing requirements). Floyd suggests that prototypes should be expanded into the target system or integrated into it — that is, the prototype is an earlier version of the final product. This implies using similar production tools for the prototype as for the final deliverable and thinking about modularity, both of which may require more time and expertise than the "quick and dirty" prototypes

Approach	Purpose	Topic of Investigation
Explorative	Elicit requirements, determine scope and different alternatives of computer support	Requirements
Experimental	Try out technical solutions to meet requirements	Particular solutions
Evolutionary	Continually adapt a system to a rapidly changing environment.	Evolving requirements

Table 2.1: Three purposes of prototypes according to Floyd [79] (table redrawn from Schneider's summary [220].)

advocated by designers, which are created with the expectation of being discarded.

2.2.4.2 Prototypes as Immature Products

Riddle [212] states that "prototyping is an approach to software development that emphasizes the preparation of immature versions that can be used as the basis for assessment of ideas and decisions." Riddle identifies two "dimensions of immaturity" along which a prototype may fall short of complete software: a prototype may offer less than a final, polished system in terms of *quality* (response time, maintainability, robustness), or in terms of *functionality*. While prototypes should be produced quickly, Riddle also stresses that a rational, controlled approach to prototype development is needed to preserve modifiability and understandability of the produced code, which suggests that the implementation of the prototype should be integrated into the main production codebase to some degree. Finally, since prototypes are constructed for assessment, Riddle argues that tools should also provide ways to instrument prototypes to gather pertinent usage data automatically.

2.2.4.3 Presentation Prototypes, Breadboards, and Pilot Systems

Lichter et al. [166] present case studies of prototype use in industrial software development and introduce a taxonomy that distinguishes kinds of prototypes, goals of prototypes, and prototype construction techniques. Four different kinds of prototypes are distinguished, based on the phase of software development they support:

1) A *presentation prototype* is used as a persuasive tool to convince a client of the feasibility of a project before starting major work on it. Other authors also describe prototypes as persuasive tools, but usually as the outcome of some design process, not its precursor.
2) A *prototype proper* is a "provisional operational software system" that is limited to specific parts of the user interface or implementation.
3) A *breadboard* is designed to clarify implementation problems for the development team and does not usually involve end-user feedback.
4) A *pilot system* is any software not constructed specifically for experimentation or communication, but part of the core project being developed (e.g., an alpha version).

The purposes of prototyping are adapted from Floyd (exploratory, experimental, and evolutionary). Construction techniques are distinguished based on whether functional coverage is horizontal across application layers (e.g., user interface only, database only) or vertical (e.g., implementing all aspects touched by the shopping cart in an ecommerce

system). Lichter et al.'s review of five real-world case studies showed little consistency in the selection of prototyping strategies in the surveyed companies.

2.2.4.4 Capturing and Sharing Knowledge Gained from Prototypes

Schneider [220], in investigating the role of prototypes in software engineering, lamented that frequently, no systematic effort is made to capture and share the knowledge gained from developing and testing prototypes. Because prototypes only examine particular details of a future product, they often cannot stand alone and require their developers' explanation to clarify context and scope: "The prototype itself is not well suited to indicate what it does well or poorly". Schneider therefore argues that the right level of analysis is the "developer-prototype system" since only the two together can fully capture intent and meaning. Documentation for each prototype should thus be systematically captured through design tools.

2.2.5 SYNTHESIS OF THE SURVEYED MATERIAL

Given the previous review of both human-computer interaction and software engineering literature on prototyping, we can now combine the various presented perspectives in to a single framework that addresses purpose, aspects, and functionality of user interface prototypes. For this dissertation, we will define a user interface prototype as *a concrete artifact that can be experienced by a user as if it possessed some or all of the interactive qualities of the envisioned interface, constructed for the purpose of generating feedback*.

Three high-level goals why designers prototype have been presented (Figure 2.4): First,

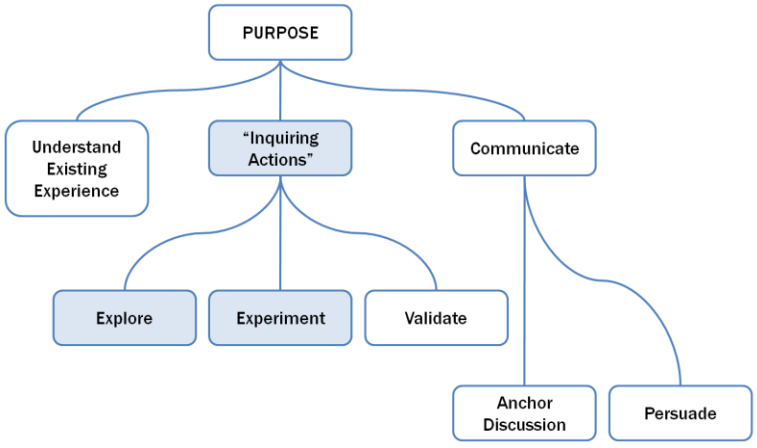

Figure 2.4: Why are prototypes constructed in design?

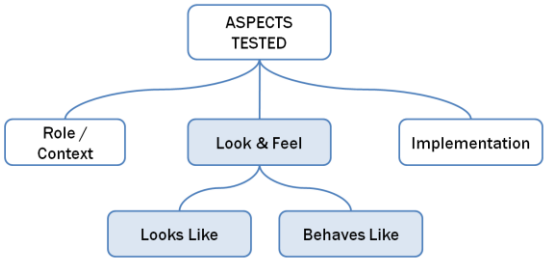

Figure 2.5: What aspects of a product can prototypes approximate?

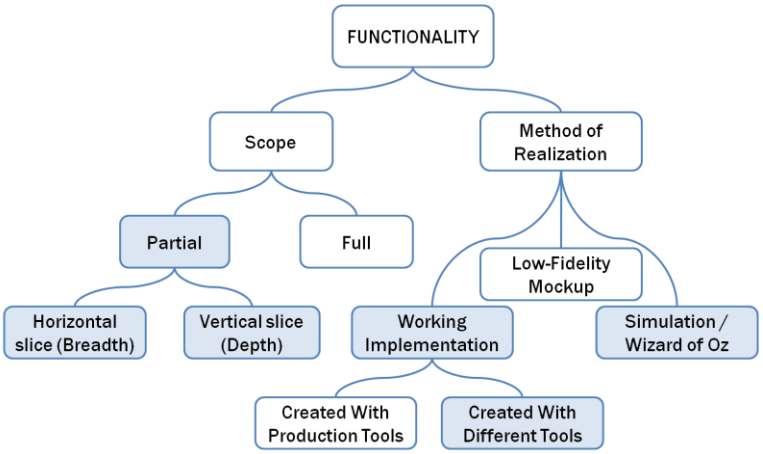

Figure 2.6: What kind of functionality can prototypes exhibit?

prototypes are built to give the designer experiential insight into some situation that already exists [52]. Second, prototyping is a technique to gain information about possible future situations [84]. As described by Floyd [79] and Haenlein [100], this stage of prototyping can have three different goals: to explore the space of alternatives, to conduct more focused experiments comparing two or more options, and to get real-world validation. Third, prototypes are used to aid communication between different project stakeholders with different "languages." Within a design team, experts with different realms of expertise use prototypes to serve as boundary objects [233] that can bridge language differences and serve as a common referent in discussion. For communication with clients, prototypes are frequently constructed to persuade the client.

Three different aspects of a final product can be tested in a prototype (Figure 2.5), as described in Houde & Hill [126]: The role a current or future product plays for a users; the look and feel of the product, and its implementation strategies. Within the category of look and feel, designers further distinguish between "looks like" prototypes that express the

aesthetic, visual, and material qualities of a product, and "works like" prototypes that exhibit interactive behaviors.

Works-like prototypes can either exhibit full functionality, or limit functionality by selecting a horizontal or vertical slice of behavior (Figure 2.6). The functionality in a works-like prototype may or may not share implementation strategies and tools with the final product. Thus, four different realization methods are possible: building a working implementation with the same toolset as the final product; building a working implementation with a different toolset specifically geared towards prototyping; creating a lower-fidelity approximation; or simulating the functionality.

The prototyping tools in this dissertation support the creation of a specific subset of prototypes (shown through shading in Figure 2.4–Figure 2.6). The introduced tools focus on prototypes created to explore design options or test specific ideas through experiments; these prototypes have working interactive behaviors, but are not necessarily comprehensive and are expressed in a new, prototype-specific tool, rather than in production-ready code.

With this particular point-of-view established, we next review related prior research into authoring techniques and systems.