Chapter 1: Design

Everyone is familiar with the word design, and everyone has an intuitive understanding of what it means. When we see a majestic building, we know that it was designed before it was built. When we see an especially striking car, we might admire its excellent design. When we encounter an everyday product that is difficult or unintuitive to use, we might refer to it as poorly-designed.

We also know that design is an activity performed by people, often (but not always) called designers. Graphic designers, product designers, architects, urban planners, and game designers are professionals, making a living by designing things for the real world. Numerous domains rely on designers to envision and lay out a roadmap for things to be. Some designers, especially in architecture and fashion, have become famous as a result of their design work.

Software must also be designed, whether we are talking about games, financial systems supporting the banking industry, flight control software for modern airplanes, the operating system in your mobile phone or tablet, or even the homegrown applications developed by individuals in their basement or garage. Like other designers, designers of these software systems must make numerous decisions to shape the software to be created. This book is about the kinds of design decisions that must be made during the software process, and how software developers make those decisions.

What distinguishes this book from other books on software design is that it presents an unusually broad view of design. Rather than treating design as an individual component in a larger overarching process, we postulate that all of software engineering should be considered a design process. Consider the following three examples drawn from actual, real-life software development projects.

- Today's hospitals employ a range of software systems, including software for billing, patient tracking, paging, interfacing with emergency response systems, drug prescription, and operating specialized medical devices. Each of these systems operates independently, meaning that staff must enter the same information multiple times, compile aggregate information stored in different systems by hand, and manually make sure that information remains consistent on multiple systems. Not surprisingly, this can lead to mistakes and dangerous, sometimes even fatal, situations for patients […]. In response, a standard named HL7 […] was developed to promote interoperability among hospital systems.

  Gerald is a developer at a small company that specializes in software for the medical field. He leads a team that aims to build the world's first open source implementation of HL7. Using the specification of HL7 as his guide, Gerald organizes a series of team meetings to decide upon the high-level architecture of the project's implementation. During the meetings, the team discusses a variety of issues, using the whiteboard to sketch possible solutions. After much deliberation, a consensus emerges as to what the best approach is, and Gerald commits to writing a document that summarizes the decisions made during the meeting.

- Ping is software engineer at a very large company, part of a team that is responsible for its online photo sharing software. The photo sharing software is not part of the core business of the company, and was initially developed as a side project. It has always been free, and has gained a massive following; at this moment in time, it manages petabytes worth of photos.

  Because of the vast user base and the emotional value that people attach to photos and the memories they capture, the team can ill-afford to ever deploy a faulty version of the software. The team has therefore adopted a test-driven approach to development: before any new code is written, new test cases and created and existing test cases are updated. The resulting collective test suite precisely captures the desired behavior of the future version of the system: they delineate what features are being added, changed, and deleted; how the various features should perform; and how other parts of the system are affected (or not affected). As a result, Ping and the other developers on his team spend many hours carefully working out the details of their test cases.

- A theater troupe is sponsored by a major healthcare provider, and as part of this sponsorship is required to provide detailed information regarding the venues where they appear. Traditional scheduling software does not support the collection of such details, though, so they hire Chris's company: a home-based information technology consulting company that, among other things, specializes in creating custom Filemaker Pro solutions. They hope that Chris will be able to provide them with a software program that will meet their specific needs.

  Chris begins by meeting with representatives from the theater troupe and the healthcare provider in order to understand the basic requirements of the system. Early in these meetings, the representatives do most of the talking, as they explain their needs. Over time, however, the nature of the meetings changes to be much more conversational in nature, with Chris and the representatives engaging in back-and-forth discussions about how the program should behave, and what features it should provide. These discussions lead to a variety of innovative features that neither of the parties would have thought of on their own. Today, the system has been implemented and is meeting the needs of the troupe and the healthcare provider.

While a more traditional view on software engineering might label these three examples as design, testing, and requirements, respectively, doing so introduces an artificial separation and ignores that they all share a crucially important trait: in each of the three examples, people are making decisions regarding the exact nature of the software system to be created. They are making *design* decisions. Moreover, while there are differences in the settings, decision-making processes, and outcomes, the activities in which the participants engage are all thoughtful, creative, collaborative, and fluid – key qualities associated with design.

Many other activities that software engineers undertake are similarly design-like. Whether we storyboard a user interaction, create a specification, recover and subsequently refactor an underlying architecture, or create a prototype to test an algorithm, we are making decisions that influence the shape of the software system under development. We are *designing*.

To further develop the view of software engineering as a design discipline, and thereby examine the activities in which software engineers engage through the lens of design, it is important to move from the intuitive notion of design to which we have appealed thus far to one that is properly grounded in the theory of design. To this end, the first part of this book examines design from three perspectives:

1. *The abstract concept of design.* We first will develop an understanding of what it means to design, in a theoretical sense. Why do we design? What goals should a designer keep in mind? What characterizes design work? Who plays a role in design? Why is design so hard? These and other, similarly fundamental questions are addressed in the remainder of this chapter, which focuses on precisely defining design, and the next, which demonstrates why design is such a non-trivial endeavor.

2. *The artifacts of design*. The human mind is powerful, but when we are designing, just thinking is not enough. Physical representations of ideas, whether they are notes, sketches, documents, models, or prototypes, play a key role in our ability to design. Chapter 3 discusses the ways that design artifacts support designers and presents key factors one has to take into account when choosing specific representations to use in the design process.

3. *The activities of design*. Designers can choose to engage in many different design activities in addressing a design problem. They may brainstorm, meet with the client, study potential users, organize focus groups, evaluate different alternatives they have devised, and so on. Indeed, part two of this book presents an extensive portfolio of techniques that a designer may bring to the table when faced with a software design problem. Chapter 4 sets the stage for this array of techniques by providing a context for understanding the things we actually do when we design.

As we discuss design from these three perspectives, we will draw upon examples from many different design disciplines and relate them to software engineering. This will serve to place software engineering in an interdisciplinary perspective, highlighting how it is similar to other design fields, yet still presents its own unique challenges. As with the examples above, where possible, we will draw upon examples from real-world projects involving real designers.

Defining Design

Thus far, we have described design in informal terms. We have alluded to design having goals such as laying out a roadmap and shaping the thing to be, involving activities such as coming up with ideas and making decisions, and exhibiting traits such as creativity and fluidity. These all capture aspects of design, but do not define what design is. A more rigorous consideration of the concept of design is needed to anchor our argument as to why software engineering is a design discipline.

To develop our definition of design, it is useful to briefly review some of the definitions of design that have been suggested over the years, including:

1. “Decision making, in the face of uncertainty, with high penalties for error” […].

2. “To choose the things we use shall look as they do” […].
3. “A creative activity—it involves bringing into being something new and useful that has not existed previously” […].
4. “Relating product with situation to give satisfaction” […].
5. “The imaginative jump from present facts to future possibilities” […].
6. “To form a plan or scheme of, to arrange or conceive in the mind, … for later execution” […].
7. “To initiate change in man-made things” […].
8. “To plan or intend for a purpose” […].

There is great variation among these definitions, and if one were to look for additional definitions, more variation would arise still. But in considering this list, a number of themes emerge:

1. *Design focuses on identifying a novel envisioned future* (Definitions 2, 3, 5, 6, 7). Design is about what is to be, taking as its starting point what we know and have today and developing a vision for something new, whether it is a new teapot, bridge, dress, advertising campaign, painting, or software system.
2. *Design involves deliberate decision-making and planning; it is not simply acting out of impulse* (Definitions 1, 6, 8). Designers have something in mind, and what they have in mind is as much designed as how they intend for it to be achieved. Simply moving a chair in the living room is not the same as sitting down, evaluating the current furniture arrangement, and forming a plan for improving the flow of traffic while maintaining an aesthetically pleasing and functional room.
3. *The design decisions are consequential: there are stakeholders who must be satisfied with the result.* (Definitions: 1, 3, 4, 8). Designs ultimately are judged, explicitly or implicitly so, by those who are affected by it. The builder of a bridge, the major of the city that commissioned it, and those who drive their cars and bicycles over it every day, all will have opinions whether the design of the bridge is a success or failure.

Bringing these three themes together, the definition of design we will use in this book is as follows:

> To decide upon a plan for a novel change in the world that, when realized, satisfies stakeholders.

In the below, we will unpack this definition with the help of Figure 1.

Designing versus Making

Central to our view of design is the novel change in the world. This change is the direct object of design, the eventual physical manifestation of what a designer envisioned. It is also the tool through which a designer aims to achieve the underlying objective of a design project. An advertising campaign, for

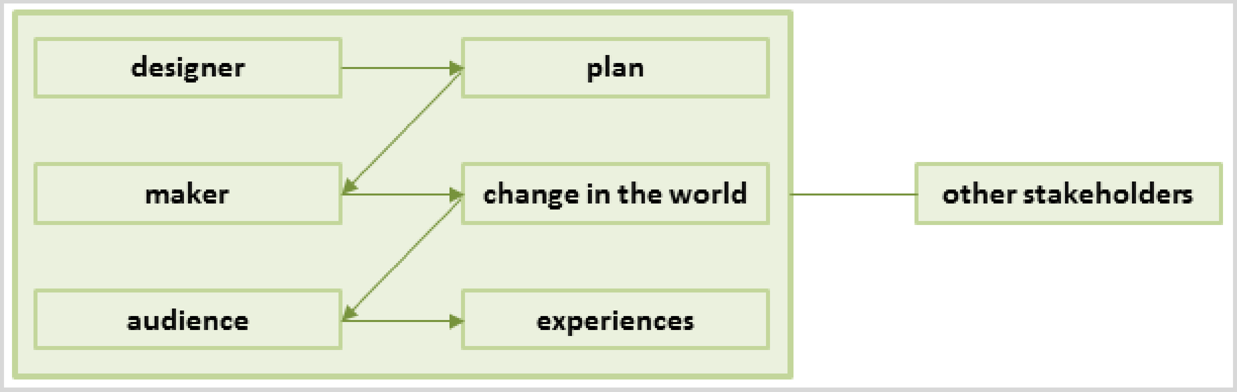


Figure 1. The context of design.

instance, is typically created with the goal of increasing sales. The design of such a campaign could involve settling upon a particular audience, identifying the key message for this audience, crafting a visual look through which the message is to be communicated, and selecting specific printed media through which the communication is to take place.

Without an effort to realize the change in the world, however, a design remains just that, a design. Someone has to sit down and produce the final artwork and layout of the advertisements, put it in a format that can be sent to and understood by publishers, and see to it that the advertisements appear in the right magazines at the right times.

This example serves to illustrate the difference between *designing* and *making*. An architect designs a building, but it is a contractor who oversees its construction as it is physically erected by a multitude of workers. The designer of a new furniture item in the IKEA product line does not manufacture it; that is left to workers and machines on the factory floor. Design, then, sets itself apart by focusing upon the plan for realization as its deliverable; a plan that specifies how the ideas embedded in the design should be turned into a tangible artifact. This plan is where designing ends and making starts.

There are exceptions, of course. For certain artists, designing and making happen at the same time and are indistinguishable. Some sculptors, for instance, are known to engage with a raw block of materials without a preconceived notion of what they will end up with. They design their sculptures while they physically chip away at a block of marble. Painters, too, will not always lead with a design that they devise beforehand, instead organically blending designing and making with the movements of their hand and the resulting brush strokes that they make.

The realities of the modern world, however, have led to a sharp distinction between designing and making that has penetrated many disciplines. Mass-produced items must be designed before they are made, especially because the final specification resulting from the design process more and more often consists of computerized instructions for some automated assembly line. Highly complex products, too, must be designed before they are made. It is impossible to build a skyscraper by laying some foundation and going from there, or an airplane by constructing a hull and then filling in the pieces. Too much can go wrong, both during construction and in the eventual use, that must be thought of and worked out beforehand.

Software is a complex product too. It also must be, and indeed is, designed. The exact nature of what it means for software to be designed is subject to debate, however. One prevailing view is that a software designers' role is to take a given set of requirements and create a design document, which describes the system as it is to be implemented at a high level. This document is then handed off to programmers, who are entrusted with writing the code that will implement the design, resulting in a program that meets the design document's specifications. In this view of software design, the programmers can be considered to be the makers of the software system.

This book embraces a different view of software engineering, one in which design extends both backwards in the life cycle, covering requirements engineering, and forwards, covering programming. The reason is straightforward: both requirements engineering and programming involve many decisions that influence the exact nature of the software that is being developed. When requirements engineers decide upon certain features, they shape the software and influence whether or not the stakeholders will be satisfied. Similarly, when programmers choose to implement some aspect of the software in a certain way, they shape the software and influence whether or not the stakeholders will be satisfied. Rather than programmers being makers, then, the compiler is the maker in our view: it translates the source code (the design as the plan for a novel change in the world) into the installable and runnable program that is the actual product to be deployed and used. This view is illustrated in Figure 2.

Audience and Experiences

For a design to be successful, it must meet with the approval of some number of people. A particularly important class of such people is its audience – those who experience the change in the world once it is realized. Often, these experiences involve usage: the kid texting his friends on a new phone, the owner of a new bicycle taking it for a climb and descent of a nearby mountain, and the grandma moving along with her new walker, all use a product that was designed by someone somewhere. The primary means through which these users evaluate their satisfaction with the product is through its utility. If the phone is awkward to hold during texting, if the bike does not have the right gear to climb a steep hill, or if the walker does not smoothly move over the surface, the experience is negative and the design underlying the product not appreciated.

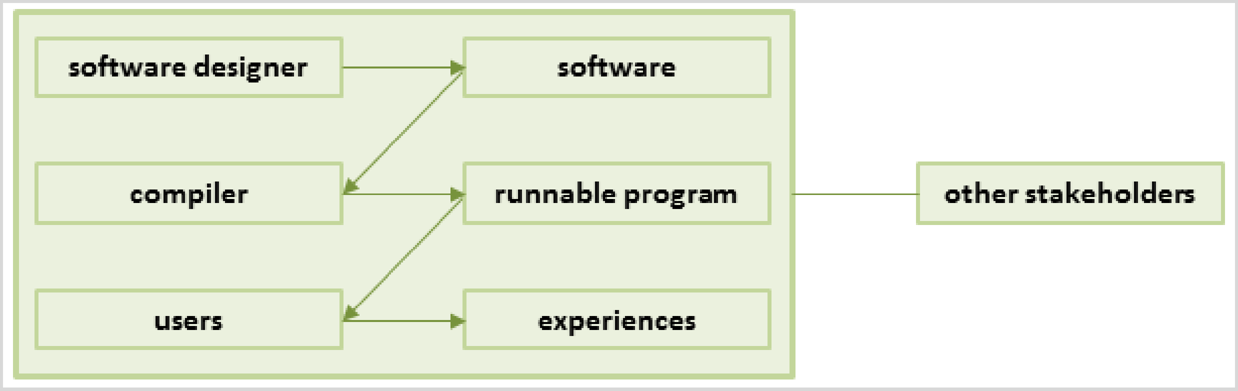


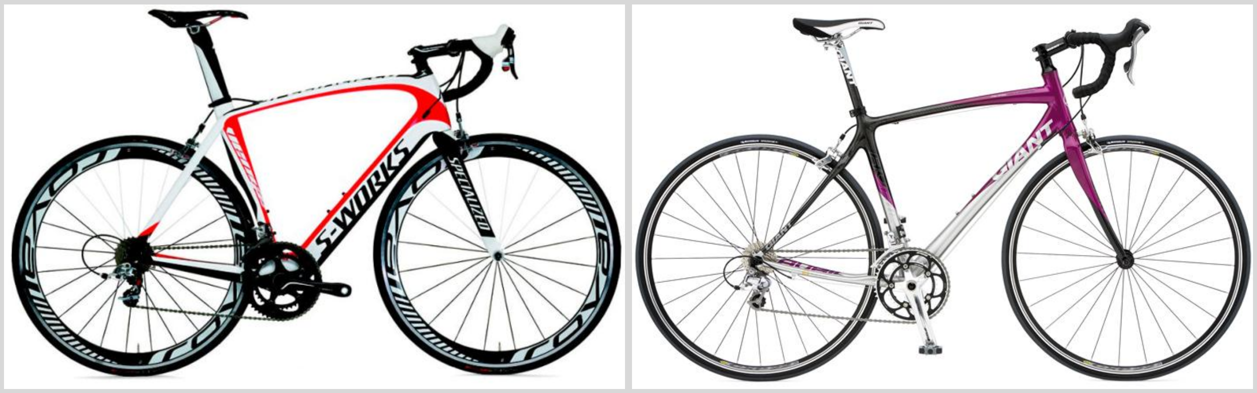
Figure 2. The context of software design.

Not every product resulting from design is experienced through usage, however. Consider an advertising campaign. Its audience does not exactly use the advertisements directly, but rather takes in the message through other means (e.g., viewing, reading, hearing). For the design of the advertising campaign to be successful, then, it must influence its audience through those means to have the intended effect of people (re)considering what product they buy, who they vote for, or what event they go to, to name a few possibilities. In some sense, when such behavioral changes occur, they constitute an implicit approval of the campaign by its audience.

Conscientious designers consider all possible ways in which a product may be experienced, and thus judged, when they are designing. The bike should not only be functional and have the right gears, but also exhibit the right amount of stiffness in its frame, the right comfort in seating, the right "zip" in its wheels, and the right aerodynamics in its handle bars. Its paint job, too, should be appealing (see Figure 3). The phone should not only make calls properly and afford swift creation of text messages, but also be up-to-date with the latest look and feel so the kid can impress his peers. Buxton provides an interesting example from his personal use of two different orange juicers, both of which easily achieve the same goal (squeezing juice out of oranges), but one of which has a much more satisfactory experience because of its ratcheted handle motion and associated sound […].

An audience is frequently not uniform in its constitution. The design of airplanes is an example. Clearly, different kinds of travelers exist. The business person who travels all the time, the casual traveler who occasionally flies to exotic vacation destinations, and the family with small children on a budget holiday, all exhibit different needs. As anyone who has set foot on a modern plane and taken a close look at the different cabins can attest, designers have taken these differences into account in the interior design of airplanes. Meanwhile, pilots, flight attendants, ground personnel, and maintenance personnel are also users of airplanes, and thus part of its audience for the designer. They all interact with the airplane and have specific needs that should be accommodated. Indeed, airplane design has such a varied audience that specialist designers are responsible for different aspects of the plane.

Software, too, has an audience that may be made up of a diverse range of users. Consider the U.S. Internal Revenue Service online tax filing system. It has to tend to a broad range of individuals, in terms of financial situation, family situation, work situation, ability to use advanced software, and so on. Any banking software has to tend to individuals accessing their accounts from home, to tellers working at



the office, to auditors wanting to examine the financial status and practices of the bank, to security

Figure 3. A high-end road bike on the left versus a less sophisticated road bike on the right. A rider will appreciate the design of the bike on the left on many different levels.

personnel wanting to protect the bank's electronic assets. Moreover, the abilities of these individuals may well vary from someone who rarely uses computers to someone who is fluent. The experiences of all must be kept in mind when designing.

Other Stakeholders

The audience is not the only class of people that a designer must keep in mind when they are working on a particular design. Most design projects involve other stakeholders: groups of people who might be affected by, or have a say in, the change in the world that is to be designed. The design of an automobile for mass production, for instance, will have to take into account its audience, which includes car owners (who may have wishes regarding its use, space inside for others and for transporting items, look, overall performance, and so on) and mechanics (who will need to service the car and thus be able to easily diagnose, access, and if needed replace various components), but also the following stakeholders:

- The car company's executives, who have a vision for their brand, who want to make sure the automobile will make them money, and who want to make sure it will not garner them a bad reputation as a result of mishaps or quality problems.
- The legislature, who has set rules regarding air pollution, minimum mileage, and safety standards.
- Assembly line workers, who, even though they are supported by powerful robots these days, must be able to swiftly, precisely, and reliably put together the car.
- Automobile magazine reviewers, whose opinions may well sway the audience into buying, or deter them from buying, through the articles they write.

None of these people will directly use the car once it is produced (unless they buy the car, in which case they are part of its audience as well). Yet, an awareness and understanding of all of these stakeholders' expectations, habits, biases, perceptions, and opinions is crucial if a designer is to succeed. Designing a car that owners love, but is too expensive to manufacture, is as equally a failure as designing a car that is cheap to manufacture but owners loathe. In both cases, one of the parties will be extremely dissatisfied and the car will not last as part of the company's portfolio.

In general, designers must weigh the importance of various stakeholders. What if your client wants you to create a product that would be potentially dangerous to its users? What if you can address all of the considerations that are raised by future users, but the product will be quite a bit more complicated to manufacture? What if the legislature recommends, but does not require, meeting certain objectives; do you meet them if it means foregoing some features that would make it stand out to users? There are countless variations on these questions, and part of being a successful designer is knowing how to balance them.

It is important to consider all stakeholders in the design process, which sometimes means that one has to consider adversarial stakeholders. Architects designing a bank must consider security features to thwart bank robbers, and the designers of financial software must think of how to fortify it, not just against undesired access by hackers but also against ill-inclined employees who may be tempted to steal money.

A final stakeholder is the designer themselves. The designer stakes their reputation, as well as the reputation of the design company they work for, on the success of the design. After all, it is their creativity and skill that are showcased by the final design. In cases where stakeholders' desires are at odds with the designer's preferences, the designer has to decide how to negotiate the differences, and when to concede.

Software, of course, also involves stakeholders whose experiences must be taken into account. Some software is explicitly commissioned, built to order for a specific situation faced by a client, making the client a major stakeholder in the process. Other software is sold by the same company that designs it, in which case the customers' preferences are more directly prioritized. Yet other software is developed in house, or perhaps created as a base software package that is customized for different clients. In each of these cases, a different balance of stakeholders' needs must be achieved. Competitors, standards bodies, reviewers, legislature, and undesired users, of course, must also be considered in software design.

Designers and Plans

Returning to our definition of design, to decide upon a plan for a novel change in the world that, when realized, satisfies stakeholders, it is up to the designer to create this plan. The plan is often called the final design, or simply design, and emerges as a set of blueprints for constructing a building, detailed schematics in engineering design, or models and drawings with measurements and material choices in product design. The form in which a final design is expressed, thus, can vary, a topic to which we will return extensively in Chapter 3.

An important property of the final design is that it should be sufficiently precise such that, when followed, it leads to the change in the world that the designer envisioned. If the final design is too ambiguous, additional decisions must be taken during the realization of the plan, and those decisions may or may not be consistent with the vision of the designer and thus may or may not work towards the desired change in the world. A house with an entertainment room for which the power specifications are not worked out may just end up with the contractor not leaving sufficient outlets or not leaving them in the right spots.

At the same time, of course, a designer does not want to waste valuable time specifying rote instructions. They must be able to count on the experience of those following their plan to complete the work within the vision established. The longer, indeed, a designer and maker work together, the more the designer can count on the maker knowing what is needed. It is not unheard of situations where makers spot and help correct problems with a designer's design upon receiving it.

In some cases, design does not end until the actual change in the world is complete. Consider landscape design. A designer will first create a plan, typically as a sketch or annotated diagram of where the plants, flowers, trees, rocks, and walkways will go, that is shared with the client and discussed. Upon the actual creation of the garden, however, it is not uncommon to see additional interaction between the designer and client, moving items around, changing the plan on the spot, and sometimes even redoing aspects in quite drastic fashion – design is still taking place. The same happens in fashion design. A typical fashion designer will make rough sketches that are interpreted by skilled and trusted employees to create the actual garments. Yet, when it comes time for the runway show, the fashion designer is back into action, sometimes making lots of adjustments at the last minute to make sure the garment is just right. In these cases, the final design, or plan, is implicit, embedded in the artifact that is created. Often, one sees this with "one-of" products, where each instance is unique (as with the example earlier of the sculptor for whom designing and making are indistinguishable, both ending when the sculpture is done).

This leads to the question of who, then, the designers are in cases like this. Clearly, the landscape architect and the fashion designer are designers; they are responsible for much of the vision of the change in the world that results. But in discussing and adjusting the garden with the client, the client becomes a co-designer with the landscape architect, and in letting the trusted employees make a variety of decisions in the creation of the garment (see Figure 4), the fashion designer implicitly lets them be designers as well. While the client and employees typically will not be called designers, they certainly engage in design work.

The importance of this point cannot be underestimated: anyone who is performing design activities should be aware that they are doing so, and trained with a portfolio of techniques that they can bring to the table.

So it is in software. Requirements engineers, architects, project leads, programmers, testers, and others all engage at various times in design-like activities, and it is their collective mastery of design that will determine the eventual success of the final design – the software to be delivered and ran. This book, then, documents a set of design techniques that software designers, broadly speaking, can bring to the table when faced with the creation of a new piece of software.

Design Disciplines

Design takes place in many domains. Well-established design disciplines include architecture, urban planning, graphic design, product design, game design, automotive design, furniture design, fashion design, landscape architecture, theatrical design, sound design, floral design, industrial design, typeface design, instructional design, book design, and communication design. In all of these, the designer must decide upon a novel plan for change in the world that, when realized, satisfies the stakeholders.

Yet, not every designer's job is the same. Each discipline has its own kinds of problems, its own set of typical stakeholders, and its own set of design methods to address its design problems. The discipline in which a designer works, then, shapes the way they think about design. A fashion designer thinks of their work very differently than an airplane designer, and an urban planner sees their work as quite different from that of a furniture designer. Each of them practices design, but each lives in a world of very different concerns and very different decisions that must be made.

A useful way of illustrating these differences is to organize the design disciplines along a spectrum according to the qualities they prioritize: is the priority aesthetics or function? A designer who is to create a floral arrangement for a wedding is primarily concerned with pleasing the bride and groom with the overall aesthetics of the arrangement. Contrast this with a designer who is to design an automated assembly line that packages different kinds of cereal into various size boxes. The primary concern here is reliability, as each time the assembly line breaks down or must otherwise be serviced represents a loss of productivity. To the operators, the effectiveness of the assembly line is much more important than whether it looks pretty.

Neither kind of design is easy. Aesthetic considerations are inherently subjective, and success can vary greatly depending on the particular audience to be pleased. By comparison, functional considerations can often be modeled in more certain terms, with equations and numbers standing in place of fickle human opinion. However, functional concerns carry harsh penalties for failure. A painting may not be quite what the client envisioned, but may still be "good enough". Functional design products that do not work are often simply useless, and may even be dangerous. These are not hard and fast rules, of course, but serve to illustrate that both ends of the spectrum present challenges to the designer.

Quite a few disciplines demand that designers consider both the aesthetics and the functioning of the change in the world they are to design. Architecture, product design, and automotive design are a few of the disciplines that rely especially heavily on both aspects. Of course, the degree to which both must be taken into account may vary. A Ferrari is carefully designed to please its owners auditory, visually, olfactory, and tactilely, without sacrificing any aspect of it being a top-notch engineering feat. The Tata Nano, on the other hand, is designed to be cheap and functional. The driving experience is secondary: it is adequate, but certainly not optimized.

Software design is another domain where aesthetics and function are often both high priorities. As with other disciplines, the degree varies by the software that is being designed. Most games rely on a careful marriage of engaging graphics and robust functionality to create a satisfying experience. Tax software must be error-free, but also must present a clear, usable interface to its users. Both of these examples contrast with other software that, for instance, is created in the name of art. Such software focuses on presenting appealing visual imagery, with minimal emphasis on the rigor of the underlying code, which may well be inelegant and less than optimal. As another example, a car's electronic stability control system has no user interface; the designer can focus solely on the interactions it has with the car's various actuators and sensors.

Key Lessons

- Design is "to decide upon a plan for a novel change in the world that, when realized, satisfies stakeholders".
- Designing is different from making in most situations, though in some cases the two cannot be distinguished.
- Designers must target a design's audience; the people who constitute the audience directly experience the change in the world and must be satisfied with the experience they have.

- Designers cannot just take the audience into account, they must also take into account a range of other stakeholders that must eventually be satisfied with the change in the world and the process of getting there, including the client, key people of influence, malicious parties, and themselves.
- While a small group of people are carry the job title of designer (or equivalent terms such as architect), most project involve many more people who are involved in design work.
- Software engineering is a design discipline, one of many that exist.
- Design work takes place throughout the software development life cycle.