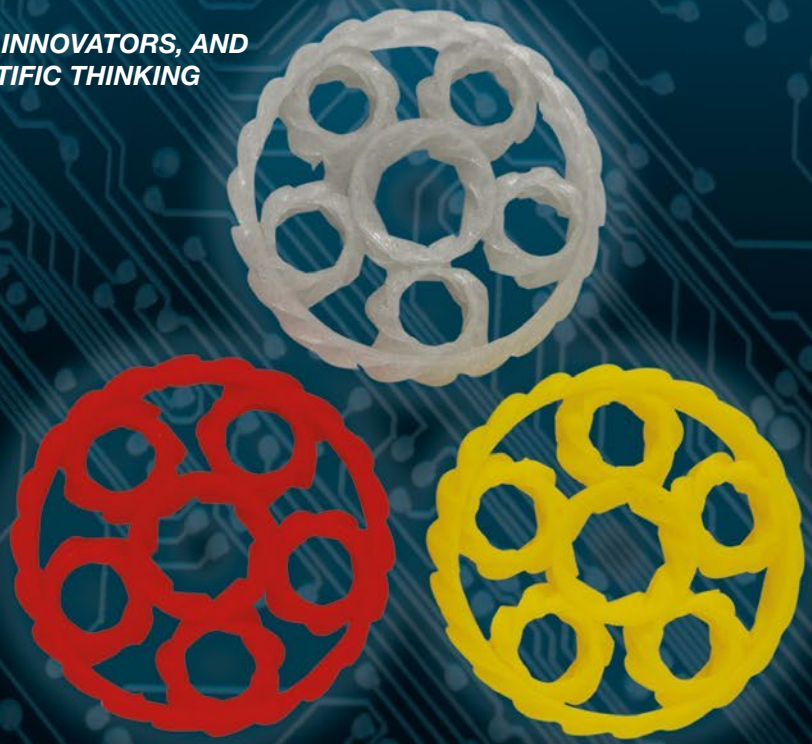
TECHNOLOGY IN ACTION™

The New Shop Class

Getting Started with 3D Printing, Arduino, and Wearable Tech

THE TECHNOLOGY, THE INNOVATORS, AND HOW TO INSPIRE SCIENTIFIC THINKING



Joan Horvath and Rich Cameron
Foreword by Coco Kaleel, Mosa Kaleel, and Nancy Kaleel

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

Arduino. 3D printing. Wearable tech. What *is* all this stuff? If you are a parent, teacher, or school administrator, you may be aware of a wave that the young people in your life are riding, but you may feel like you are caught in a riptide of terminology and being towed farther and farther from land. As technologists working in this sphere, we became aware of many people who felt like you do (and we got tired of answering the same questions many times). To try to make information available more broadly than we could in person, we have written this book to answer your critical questions. What does it cost to get started with these technologies? What do I have to learn to get started? Beyond that what will I (or my kids) learn by taking on these challenges?

The technologies we talk about in this book for the most part arose out of a do-it-yourself, "hacker" or "maker" culture. This culture (which you will read more about in Chapter 1) frames learning as something you do yourself, usually online or by making things with like-minded people. A disconnect between this culture and traditional education has developed. The authors are a traditionally educated aeronautical engineer-turned-educator (Joan) and a self-taught hacker and 3D-printer expert (Rich, known online as "Whosawhatsis"). In this book we come together and explore the gaps and similarities in our world views. Through our partnership, we try to show a model of how traditional education can merge with the makers and hackers of the world to create a much richer learning experience than is possible to have by learning passively.

Chapter 1 gives you an overview of the difference in mindset between the two of us and provides a road map for the rest of the book. Chapters 2–4 go into some detail with regard to some of the basic technologies: an easy-to-learn microprocessor called an Arduino, 3D printing, and robotics. Chapter 5 shifts a little and talks about how people are creating spaces to learn by making things, both in public spaces and at schools. Chapter 6 talks about building on these base technologies to do "citizen science" (real science projects with general-public participants). Chapter 7 is an introduction to the world of wearable technology—creating clothing that can light up, react to the world around it, or just do things that seem like magic. Chapter 8 is an overview of some easier technologies and explains our view on why you may not want to start with these training wheels.

Chapters 9–11 take a step back to talk about the cultures that grew these technologies. Chapter 9 gives you some insight into the open source world—a technology community in which everyone shares ideas and builds on them. Chapter 10 discusses how to bring girls and women into the maker community, where they are wildly underrepresented. Chapter 11 explores the case study of a community college program focused on having students make things, including a project to create 3D-printed objects for the blind.

Chapters 12–14 shift to talking about some of the motivation for bringing a maker style into a classroom, including the fact that it is a good way to encourage students to become scientists. These three chapters discuss how scientists actually work and think and tell stories of how many of them came to science through a love of taking things apart. You may see some of your young makers in these stories.

Finally, Chapters 15–17 bring it all together and discuss how the other chapters all bring evidence that some of the best learning comes through actually creating something with your own hands, arguing that this is a particularly effective way to learn science.

Before all that, though, we start with a foreword by some of the friends who inspired us to write this book. Coco Kaleel came into our lives when she was about 11, and we were working at a 3D-printer company. Her parents were not technologists, and they desperately needed a guide to all things maker. They will tell you about their journey, and we hope we can make yours easier than theirs was!

Use this book as a starting point to guide your own explorations or those of a young scientist in your care. We have tried to give you pointers to many other references without being overwhelming. Of necessity, this means we have made choices about what to include and what to leave out. There are many other ways to do most of the things here, and we made the choices we thought opened the most doors. We only ask that you start stepping through those doors and out into new worlds that you will help create.

PART I

The Technologies

This first section of the book introduces you to Arduinos, 3D printing, and the mindset of the community that has developed the consumer versions of these techologies.

Chapter 1 is an overview of the differences in mindset and views of learning between the maker/hacker community and the traditional, educational one. It is also a guide to the rest of the book. In Chapter 2, you learn about the Arduino microprocessor and the low-cost ecosystem of sensors, motors, and other electronics that has sprung up based on the Arduino. One of those ecosystems is the low-cost, open source 3D printer, introduced in Chapter 3.

If you take Arduinos, some motors and sensors, and maybe a few 3D-printed parts, you can make yourself robots and other things that move on their own. Chapter 4 discusses the basics of robots and gives you some entry points into the overwhelming number of kits, ideas, and online tutorials.

Taken together, these chapters provide the material you need to know before moving on to the more complex applications of the technologies covered in the rest of the book.

CHAPTER 1

21st Century Shop Teacher

The words *shop class* conjure up a messy place where sawdust and metal shavings pile up on the floor as awkward birdhouses are built up on the tables. *Computer lab,* on the other hand, brings up images of white floors and walls, whirring fans, and overly-good air conditioning. It is also the last place on earth that you would want sawdust and metal shavings. School districts have been closing out their shop classes, because of perceived lack of student interest or liability concerns, as computer labs become ubiquitous.

However, a new hybrid of machine/wood shop, computer lab, and electronics bench is emerging. These are variously called hackerspaces, makerspaces, fab labs, or perhaps robotics labs. They might be spaces open to the public as a place for learning skills or using tools, or focused on some specific activity like building robots or creating fantastical costumes. They may have equipment that runs the gamut from glue guns and fabric to 3D printers, hand tools, laser cutters, and computer-numerically-controlled (CNC) machine tools. For the most part, we will use *makerspace* as the general term for this type of space, since it seems to be the commonest term in school, library, and museum settings.

When a makerspace is set up in a school, will it become the site for 21st century shop class? What will students learn there? Who can run one of these shops? If you are a teacher, how can you get past the intimidating complexity so that you can learn to use the equipment and get your students using it, too? If you are a parent, what will a home version of these spaces look like?

This chapter talks about the resurging interest in making things, enabled by the combination of low-cost 3D printing and (relatively) easy-to-program electronic components. It introduces the technologies that we talk about extensively in later chapters and what you can do with them. Finally, we introduce ourselves—a traditional engineer/educator and a hacker—and start the conversation we want to have with you throughout this book about how to reconcile these different approaches to learning and how to become conversant with what these technologies make possible.

What Is "Making?"

Being a maker is more of a state of mind than a well-defined activity. In the next section, we lay out our (different) perspectives on what being a maker *should* be and how someone should become one. For the moment, though, we will define *maker* as someone who makes something because they want to, even if they could buy what they are making. A maker also typically wants to learn how something works and learns this best by making it.

There are various levels of difficulty of making, and some are closer to fine art or crafting. In this book, we focus on the technology-oriented side of making, while recognizing that often a love of design may come from woodworking or sewing initially and then cross over into electronics, or the other way around. (Figure 1-1, for example, shows an electronic maker's foray into holiday tree design.)

Figure 1-1. *A maker's holiday tree of wooden dowels. Courtesy of Luz Rivas*

Even narrowing down making to the technological options leaves an overwhelming number of different possibilities. You may have had the experience of searching online for "Arduino," for example, and getting dozens of example of things to do with an Arduino board but no explanation of what one actually *is*. (For the record, it is a microprocessor that can control physical things, which we will meet in depth in Chapter 2.)

This book is intended to be a field guide for you to see where good entry points are for a beginner, and how to move from beginner to more advanced if you do not have a handy community around you already. In the last chapters, we talk about how making can be a good route into learning science, technology engineering, and math (STEM) subjects.

■ **Tip** If you live near a public makerspace, it likely has beginner classes (try an online search for "makerspace" and "hackerspace" plus your city name). Call them up and tell them your situation. For example, are you a parent with a kid getting interested in these technologies? They are likely to know about resources that are available regionally. If you do not live near one, search online for forums (see Chapter 9's discussion) and post about what you are trying to do. You will usually find someone willing to help, even if that person happens to live on the other side of the world.

Who Is a 21st Century Shop Teacher ?

One of the challenges of starting up a makerspace is finding people to run it. It requires a mix of skills that are rarely found in one person—a combination of comfort with traditional shop class methods plus electronics plus competence in computer programming. If a school's IT department is asked to set up a makerspace, they may not have any experience with the issues that arise with making physical things. On the other hand, the shop class or art teacher may not have a lot of experience with the computing aspects of these new hybrid skills.

The authors (Joan and Rich), shown in Figure 1-2 at New York Makerfaire, came into this space on very different trajectories. We worked together for a time at a small 3D printer manufacturer. Now we collaborate on figuring out how to teach just about any subject through hands-on creation of physical objects.

Figure 1-2. *The authors at 2014 New York Makerfaire. Apress PR photo*

Joan is a traditionally educated baby-boomer aeronautical engineer with a strong computing background. She came into the maker world in early 2013 with almost no hands-on electronics or shop experience. Rich, on the other hand, is a millennial, self-taught electronics hacker and 3D printer guru who has been involved in open source 3D printing since its earliest days (around 2008). This makes him the old-timer of the two of us. In this book, we will give you both sets of insights about how these two communities can best work together. The next section gives you our two first-person views about this community, and how we each think about traditional education and hands-on making. We want you to feel like this book is a conversation with the two of us that will help you figure out how to navigate this new world.

Joan: An Engineer and Educator Meets Making

I learned engineering from university classes at MIT and UCLA, culminating in a Master's degree in engineering from UCLA. For 16 years I was an engineer at Caltech's Jet Propulsion Laboratory (JPL), which makes spacecraft that go to other planets. In those environments, the critical skill is being able to learn things quickly. I am used to learning things top-down. Usually a first introduction would be from a book or manual, with a lot of equations. In a professional engineering research environment, particularly in aerospace, people become very specialized. I am primarily a software person, and it would have been unthinkable for me to plop down in a spacecraft electronics assembly area and touch *anything*. In the course of 16 years, I was probably in the same room with actual flight hardware two or three times, if you do not count looking down on it from a glassed-in viewing gallery.

Like most people at JPL, I always had a special relationship with the robot spacecraft I worked with. They almost felt like children, or coworkers. I worked on software that told the Magellan spacecraft what to do. Magellan was the first spacecraft to create radar images of the surface of the planet Venus, which is covered with dense clouds. The software absolutely, positively could not have bugs.

We had to be very creative and deal with situations that no one had ever really thought about before. After all, how many people think about what happens in Venus orbit every day at work? Despite that, excruciatingly careful planning and fanatical attention to detail was also necessary, and JPL in the 1980s and 1990s was probably the last place in the solar system one would exercise a "let's see what happens" maker mentality. I left JPL in 2000 and consulted in the entrepreneurial aerospace world for a decade. Then I started to spend more and more of my time as an adjunct faculty member at several institutions.

I came into the maker community early in 2013 when I was looking for material for online undergraduate interdisciplinary studies classes. By this time, I was an adjunct faculty member teaching students who were training to be elementary schoolteachers. I wanted them to see science and engineering as a process of discovery, rather than as an exercise in vocabulary worksheets. As it turned out, we decided that the learning curve was too steep at the time to fit it into that particular program, although we piloted one of the first online teacher professional development classes in 3D printing.

By then, I could see the power of 3D printing and other maker technologies and joined a small 3D-printer company. It horrified my new colleagues that I was a rocket scientist who had worked on several interplanetary spacecraft (Galileo to Jupiter and Cassini to Saturn, in addition to Magellan, and some studies of things that never flew), but had not really built anything with my hands since my undergraduate lab days. I felt uneasy touching electronics, even though intellectually I knew it was all hobbyist stuff and if I messed up it would not result in the failure of billions of dollars worth of spacecraft.

Most of my new peers could design and build a consumer 3D printer from nothing but what was in their heads. Some of them (Rich excepted, of course!) did not see the point of all my formal education if I could not sit down and just build something. Specialization seemed some sort of abdication of responsibility to them, given how alien it was to think that you would focus on just one piece of a much bigger project and rely on hundreds of colleagues to know the rest. Makers prided themselves on building something all themselves and on knowing everything about it. Having been part of a team that flew a spacecraft to map Venus seemed inconsistent with the fact that I could not wire a terminal block competently. (Note to anyone over 40 taking up making: go up a half diopter on your reading glasses—some of this stuff is *tiny*.)

Learning by Doing—Are There Limits?

When it became clear to me that using these 3D printers was pretty complicated (even by recovering rocket scientist standards), I started to develop training materials to make things easier for our customers. When I tried to teach myself how to use open source 3D printers, I found a lot of detailed information scattered online. But there was almost nothing that stepped back and walked through the overall process of creating a 3D print or that defined terms and general concepts for a new user. There were user forums, but they were organized somewhat randomly around whatever order people had asked questions. They were searchable, but you needed to know what questions to ask and what terminology to use. Often that terminology was different than conventional engineering terminology.

When I asked about this, the reaction often was that the detail was there, so what was the problem? Or, secondly, that learning on my own was sort of a rite of passage—that unless I figured everything out myself, I probably was not ready to use the technology anyway. After an extensive period of pestering experienced users (particularly Rich!) and trial and error, I slowly became competent and moved on from there. However, the way I did it was very inefficient at the community level. I saw person after person spend days dragging together the same information from scattered and inconsistent sources.

I had started my career working with early supercomputers at JPL (which had about the same computer power as the $25 processors we will talk about in Chapter 2, but that's a different story). There are some similarities between 3D printing today and the early days of computing, but even then specialization in hardware or software was pretty common, and there was less of the maker expectation to be good at all aspects.

There is an old joke that says that the difference between scientists and engineers is that scientists like to be surprised but engineers hate it. This implies that engineering is the process of working largely to apply existing knowledge. However, the maker process seems to assume that makers need to discover everything for themselves and thus be engineers who like to surprise themselves.

If you ask a maker about this, they will insist that people learn better by learning everything by doing. To me, though, it seems like this philosophy limits most people to learning only what they can invent themselves and makes it unlikely they will create new knowledge. It is all about how things work, but not about general theory and bigger picture behind it. I call this type of learning *icicles*—very deep knowledge in some areas, but with gaps in between.

In the end, I wound up writing a book (*Mastering 3D Printing*, published in 2014 by Apress) with Rich in a technical reviewer role. Writing the book and structuring material for it the way I would like to have learned it brought me up to a level where I felt competent—unless something involves one of those miniscule terminal block connectors (Chapter 2).

Global, Virtual Apprenticeship

If you are traditionally educated in a technical field, you are likely nodding your head now about how "these makers" are reinventing the wheel and avoiding doing the hard work of learning math and science the usual way. You may have been resisting having a makerspace in a school because it is "just playing." But, notwithstanding everything I have just said, it's not that simple.

I know that I am a very structured, top-down learner. Given that I am a female engineer who went to school when female engineers were a single-digit percentage of most fields, I am a bit bemused by being the "traditional engineer" in this book. However, I will accept that I have been taught traditionally. Almost all technical fields are taught in a way that favors visual learners and top-down learners.

But if someone learns bottom-up, will they be better off deriving the top-level knowledge themselves? Makers learn things by working in a makerspace and hanging around others, or by doing the same thing virtually by hanging out on forums and discussion boards. But most of all, they learn by trying things and seeing what happens. Is becoming a maker a new way of having a global, virtual apprenticeship if only learning from books is not for you? Or are they a new type of artist? Or are they creating a new discipline altogether? We will try to address these questions as we explore the examples in later chapters of this book.

Not long ago, what you could learn about electronics by cut-and-try experimentation was fairly limited because of cost of the hardware, its complexity, and access to information. The critical piece that is new is the availability of sophisticated but easy-to-use electronics and tools like 3D printers, plus nearly infinite (overly infinite?) web-based information. As we will see in later chapters, these new electronics were designed with either students or hobbyists in mind. Some of the most powerful uses of these new, accessible electronics are *in combination with* 3D printing. Both students and professionals who need to prototype or make one-of-a-kind things quickly (like product designers, or scientists, or artists) can very quickly and relatively inexpensively turn out a pretty sophisticated first version of an electronic device or a scientific instrument or a piece of kinetic art.

Many students learn best if they can immediately apply what they have learned to something concrete in front of them. Some of these students are ones who might have excelled in shop class when that was an option for them. (As mentioned, most shop classes in the United States are being shut down, for perceived lack of interest or liability reasons.) Others are fascinated by the virtual world and come into making from the programming side—also an area that is not supported well in all school districts. What does maker learning look like?

I will now hand this over to Rich to describe his path into making, and the hacker-versus-maker approach. There he is in his typical work environment in Figure 1-3.

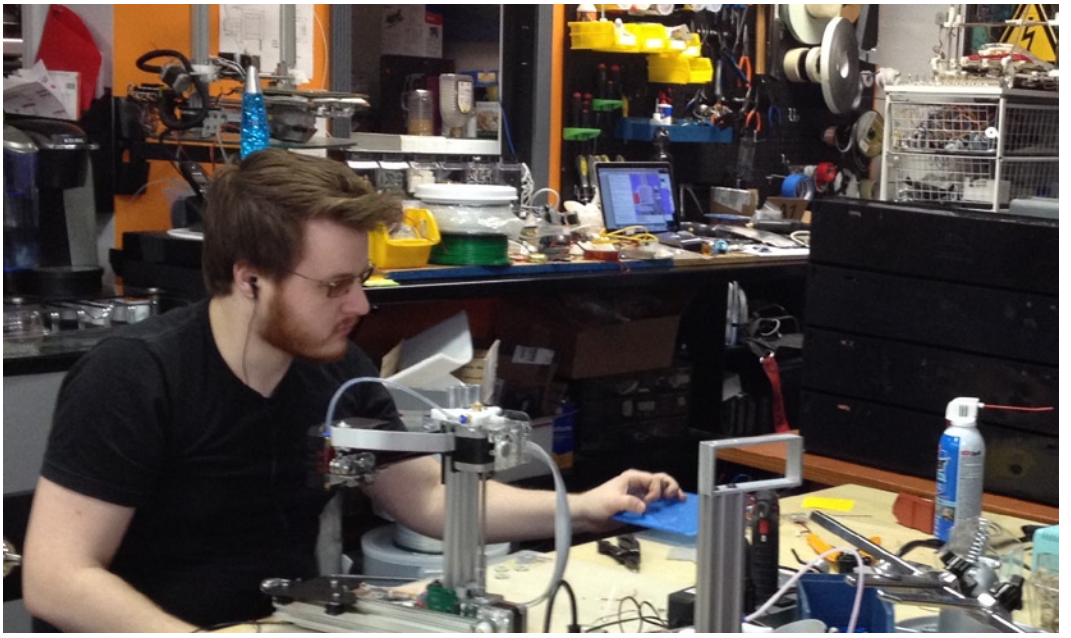


Figure 1-3. *The hacker in his element*

Rich: The Hacker Path

I learned engineering from the single greatest repository of knowledge humanity has ever produced: the Internet. Like many hackers, I am an autodidact. In school, I excelled at tests but still only passed some of my classes by the skin of my teeth. I couldn't stand to waste time on assignments intended to teach concepts through mindless repetition when they were clear to me when they were first introduced. Instead, I liked to spend my time learning C and similar programming languages to develop various software projects.

I've always found that the best way to learn anything is to first have some project that the knowledge is required to accomplish. Of course, unfettered access to information is crucial, but the Internet makes that easier than it's ever been. With the exception of one basic electronics class and the standard set of math and science classes where the students complained about needing to learn things they'd never use, I haven't gone to school for any of the knowledge I use on a daily basis. I developed one of the first low-cost 3D printers (Chapter 3) and became vice president of research and development at a small 3D-printer company on the strength of what I have been able to teach myself.

My basic electronics class taught me Ohm's and Watt's laws (Chapter 2) and other concepts over the course of a semester, but I learned far more during the first week that I sat down with an Arduino and something to accomplish (Chapter 2 talks about Arduinos). I started designing circuits and fabricating circuit boards, at first using things like perf board and conductive ink, then using free computer-aided design (CAD) software and mail-order prototyping services.

I began building robots with Arduinos as controllers, and when I needed more complex and precise mechanical parts than I could produce with hand tools, I decided to use one of these Arduinos to build a CNC mill to cut the shapes I needed automatically from CAD drawings. When looking for software to use with such a machine, what I found was the software for controlling open source 3D printers. A 3D printer, as you will learn in Chapter 3, is a robot that can make things, including other machines, and in many cases even copies of its own parts or improvements for itself. This quickly became more interesting than my robots or the CNC mill, and I've been working on open source 3D-printer designs ever since.

Hacker vs. Maker

Though the situation has been improving in recent years, the term *hacker* has been much maligned and misunderstood more often than not in the media and in popular understanding. Hacking does not consist of writing computer viruses, defacing websites, and breaking into computers for mischief or personal gain, though hacking is usually a necessary precursor to these activities.

One of the several definitions (and my personal favorite) for *hacker* is in the Jargon File (`www.catb.org/jargon/html/H/hacker.html`), probably the oldest and most complete reference for the terminology used by hackers. It says: "One who enjoys the intellectual challenge of creatively overcoming or circumventing limitations." Hacking uses and develops a person's creativity, critical thinking, and problem solving, the three most universally important skills one can have.

Some people do not like to use the word *hacker* to describe the types of activity in this book, because they think of the word in the sense of a *black hat* (as opposed to a *white hat*) computer security hacker. A stereotypical black hat hacker overcomes or circumvents obstacles imposed by computer security systems because they want to damage or steal something, whereas a white hat does so to find security holes so that they can be fixed. However, a true hacker's motivation for overcoming these obstacles is simply for the challenge (and possibly the bragging rights) of doing so. Whatever shenanigans they may get up to after the barriers are broken do not define what it means to be a hacker.

Only a small but sensationalized minority of the larger hacker community is involved with breaking computer security systems. The inherently constructive types of hacking we describe in this book have nothing to do with this type of hacking. The limitations we try to overcome are often simply the limits of what anyone has ever figured out how to do, or even thought possible. In this sense, almost everyone who ever invented some new technology was a hacker of some sort.

The maker movement grew out of this hacker culture as well as the do-it-yourself (DIY)/hobbyist and avant-garde art/sculpture scenes. Type **kinetic sculpture** into your search engine of choice to see some particularly impressive examples of what makers do. Although hacking is occasionally used for destructive ends, making is a constructive pursuit by definition. Though both terms are equally applicable to most of the things that people like me do and what goes on in a hackerspace (I prefer that classic term into over makerspace), *maker* is often seen in language that has been sanitized for those who may still misunderstand what hacking is all about. There is a subtle distinction that hacking is motivated primarily by the enjoyment of creative problem solving, whereas making is directed more toward the end product. In this sense I am a hacker first, and a maker second.

Learning by Doing: Overcoming the Limits

There is age-old knowledge that will always be useful, but in a field as fast-moving as 3D printing, the most important thing to learn is whatever was discovered yesterday. Traditional education can teach the old stuff, but when it comes to keeping up with new developments, you're on your own. The open source communities make this information available to find, but learning to learn is an essential skill. Just as a picture is worth a thousand words, knowing how to recognize and fill the gaps in your knowledge when you need to is worth more than a billion memorized facts and formulae.

There is one kind of knowledge that is more valuable than what was discovered yesterday, and that's what will be discovered tomorrow. Learning how to find information is critical, but you'll never contribute any new knowledge if you can't figure things out for yourself. Classically educated engineers tend to look down on what self-taught hackers like me do as "just playing" and think it's foolish to discover things for ourselves that are already known and could just be taught to us. However, by reproducing past inventions and discoveries for yourself without the prior knowledge, you are also learning how to invent and discover new things.

People don't become great musicians by listening to a lot of music, but by practicing simple pieces of music before they can perform difficult ones, and although watching a lot of baseball on TV might make someone more likely to get a seat at the World Series, the players on the field started in Little League. Things that are easy to discover have already been discovered, and things that are easy to invent have already been invented, but to discover or invent more difficult things that are new to the world, you need practice discovering and inventing simpler things that are new to you. If a man learns how to make a wheel, he'll be able to get to the next town. If he learns how to invent the wheel, he might make it to the moon.

Joan likes to talk about top-down vs. bottom-up learning, but I think of my method as more of a middle-out strategy, more like ice crystals spreading out in a supercooled liquid (search for videos of that online if you haven't seen it, it's pretty impressive) from various nucleation points rather than the icicles that Joan envisions. I learn best by gathering disparate, seemingly random bits of information when I need them and then get the deepest understanding by integrating and filling in the gaps between them on my own. New bits that are close enough to something I already understand just make sense and are easy to absorb, and if a gap is too large, a quick Internet search allows me to find the bit of information in the middle until all the gaps are small enough to bridge easily.

At the same time, I like to ponder the edges of my understanding and figure out related things, practicing expanding my knowledge. Unlike the way the same subjects might be taught in a school, these new areas of thought may cross into a different subjects and back, and some of the most interesting and unique topics are ones that fall between typical class subjects, and may even be things that a traditional education would fail to cover. I sometimes spend hours at a time pondering things that nobody really understands, like the connection between quantum physics and general relativity.

Physical Software

I was always more of a software hacker and never had much interest in taking a shop class when I was in school. It may seem odd then that my most well-known contributions to the open source 3D-printing community are hardware projects: 3D-printer and component designs, and other printable objects. The fact is, the tools of digital fabrication turn hardware and mechanical designs into a software problem. CAD software allows circuits, components, or entire machines to be designed and sometimes even simulated in software before any of the physical parts are made. Then the computer-controlled machines can turn those designs into physical products with minimal human interaction.

These 21st-century shop tools aren't yet the simple IT devices that 2D paper printers are (though some less honest 3D-printer manufacturers make them out to be), but they're a lot closer to it than the human-operated machine tools that they replace, and they're getting closer. This fact made it possible for me to do hardware design and fabrication within the software realm that I was comfortable in, allowing me to think of hardware design as a physical extension of software hacking.

The first CAD program I learned to use was CadSoft EAGLE, a program popular in the Arduino community for designing circuit boards. I taught myself how to use it to design my own Arduino-compatible development boards and robot controllers. Then I uploaded my designs to online PCB prototyping services so that I could order my custom boards and receive them in the mail. Once they arrived, I would solder in the components, try the circuit, and (if necessary) modify the design and re-order.

The tool that enabled me to use software to create real things the most was OpenSCAD. OpenSCAD is the quintessential "physical software" tool, billing itself as "the programmer's solid 3D CAD modeler." In OpenSCAD, you build up complex 2D and 3D objects from simple primitives in a process called *constructive solid geometry*. To do this, you write code in a language with a C-like syntax (which my prior programming experience allowed me to pick up in a matter of hours). This approach to design isn't for everyone, and there are many more mouse-oriented CAD options, but for someone with a software hacking background like mine, it's ideal. I can quite literally code physical objects the same way I would code a computer program.

How the Paths Merge

So who will be a 21st century shop teacher? Our answer is that it will take people like the two of us coming together to create bridges between the traditional education and the maker communities. Those of us who know book-learning science will continue to pass it on. But for relevance and application, 21st-century shop will need a big dose of actual making things. As Rich says, much of what we know now did not exist a few years ago, and learning to learn will be the high-value skill as many barriers to prototyping and manufacturing fall.

However, it is also necessary to learn accurate material. Currently the maker community manages this by being small and an everyone-knows-everyone type of group, but of necessity this is changing. Not everyone has the ability to recapitulate Isaac Newton and other greats to reinvent everything as they go, either.

Given that, how will people learn five or ten or twenty years from now? As Joan found when she tried to learn 3D printing from unstructured materials, even a very good technical education does not necessarily make it easy to learn a whole new field from scratch. However, it did help her organize what was known to make it easier for everyone who comes after.

To take that to the next level, a much closer collaboration between educator and hacker is required, and the result is this book. We argue a lot and do not see entirely eye-to-eye on the best path for education. However, we have mutual respect and can see we each learned best in our own ways. We also have a shared love for plain cheese pizza, which helped create common ground in the beginning and now is just a plus.

■ **Note** Industrial and product design education has traditionally immersed its students deeply in the creative process and what we describe here as "hacking." If you are trying to create coursework for more advanced students in a hacker style, some seminal books in this field are from the design or psychology literature. One classic is Mihaly Csikszentmihalyi's *Creativity: Flow and the Psychology of Discovery and Invention* (HarperCollins, 1997). Csikszentmihalyi is best known for his concept of "flow"—a state in which people are working right at the upper limits of their abilities and are very happy and productive because they are learning and pushing their limits. Makers almost by definition will be in this state often. Figure 1-3 could be an illustration of the concept of flow.

Defining Your Problem

Many who are reading this book likely are parents or traditional educators. This book is designed to help with situations like the following:

- Your child has asked for an Arduino starter kit for her birthday, and you were embarrassed to discover that it was not a dog breed, as you originally assumed.
- Your principal has announced a maker initiative for your school and asked you to coordinate it and produce a budget. You have little or no idea how to proceed.
- You are a school administrator, and parents are asking what practical skills their children are learning. Or perhaps parents are asking about when you will include 3D printing and maker technologies in the classroom.
- You bought a 3D printer to teach math and science either at home or in a classroom, unboxed it, printed a Star Wars figurine, and wondered, "Now what?"
- 25 Arduino starter kits have just been delivered to your school courtesy of a donor, and you had no idea so much wire and so many fragile small parts would be involved. And no one has the least clue what to do with them or how to teach with them.
- You already are into this type of learning but need some evidence to convince dubious colleagues to introduce maker activities into your curriculum.

To address situations like these and more, we have structured the book into a chapter for each of the major types of maker technologies. Table 1-1 is a survey of what we address in this book. We also list the basic skill sets that you will need to learn concurrently if you are going to use these technologies and the chapter that goes into each area in more depth. In each chapter, we give a rough indication of how much it costs to get a "starter set" and get going with it. These skill set and costs summaries appear at the end of Chapters 2 through 8.

Table 1-1. *Typical Maker Technologies and Activities*

Technology	What It Is/Does	What You Need to Learn	Chapter
Arduinos	Microcontroller that controls lights and/or sensors, motors, etc.	Programming Wiring Possibly soldering	2–7, 15–17
Raspberry Pi	A basic computer	Linux operating system	2
Circuit stickers* LittleBits* LightUp* LEGOs*	Make circuits by drawing or magnetic attachment or other simple connectors	A bit about circuits	8
3D printing	Makes physical items based on 3D models	3D computer modeling or scanning, software that slices model into layers, physical interaction with printer	3–7, 15–17

(*continued*)

Table 1-1. (*continued*)

Technology	What It Is/Does	What You Need to Learn	Chapter
Robotics	Design/assemble robots	Wiring, programming Possibly soldering and/or machine tools	4
Wearable tech	Clothing that uses Arduino-like devices to make clothing that lights up, senses things, etc.	Arduinos plus sewing	7
Cosplay	Costume creation	Sewing and glue Possibly Arduinos	7
MakeyMakey*	Boards that plug into computers and allow anything to be an input device	Circuit design basics	8
Citizen Science	Using maker tech to do science	Arduinos, sensors, science	6, 16, 17

**Starred items have shorter learning curves, at least in the beginning.*

Part of the point of playing with these different technologies is to learn the skills listed here (versus thinking that you would need to learn programming or soldering first, for example). The learning curve can be pretty steep, as we will see when we go into Arduinos and 3D printing in Chapters 2 and 3 respectively, and it is hard to learn these skills in isolation without a project to help you focus on what to learn first, as Rich noted in his backstory earlier in this chapter. For that reason, these skills are often taught in project-based ways that teach a little bit of each of the skills needed (say, programming and wiring just a few components, in the case of the electronics-oriented spheres). You can do progressively harder projects as you build the many skills needed to get started.

■ **Tip** It can be frustrating to embark on electronics projects if you're not sure what parts you will need. One way to get around that is to buy a beginner's kit to get started. "Learn how to…" kits are sold by many vendors, notably Sparkfun (`www.sparkfun.com`) and Adafruit (`www.adafruit.com`). These companies also have tutorials on their sites for a wide variety of skill levels. Each chapter in this book will give you some ideas of what to buy to get started more specifically, but these two sites are a good place to look around in general to see what is possible.

Making a Scientist

Let's take a step back now and ask: why are we rushing around trying to figure out how to use these maker technologies in education? Making is good training to be a scientist. If you just absorb preexisting knowledge without some discovering of it yourself, you will not appreciate or be able to see it as a process. All too often, Joan encounters someone who thinks science is too hard for the average person to understand, or who thinks that the best way to teach science is as a vocabulary lesson, with worksheets that match a concept and word. This kills the idea of science as exploration and inquiry, which is what makes it fun (and hard).

Scientists, though, are usually makers. Anyone doing laboratory work might need to make or modify equipment. By the nature of their work, typically scientists are doing something for the first time. For the most part, they need to design experiments that can be done by existing equipment. More and more, though, these same low-cost electronics that allow you to learn in the first place can be used to create simple equipment capable enough to do a new type of experiment, or to collect vastly more data than was possible before. We talk about this aspect in Chapter 6, when we discuss citizen science and open source labs.

More fundamentally, though, the maker (or hacker) mindset—the *let's see what happens* attitude—is a crucial part of being a scientist. To prove that, in Chapters 12–14 we have collected a lot of short vignettes about working scientists, engineers, and mathematicians, with some explanations of their thought processes. Some stories are about the professionals as children, getting in trouble by blowing something up (or in one case putting a fork in an outlet). Others talk about the practicalities of what they do all day. All of them, though, will give you some idea of why it is a good idea to use some precious formal education classroom time to actually make things.

The final chapters of this book tie together the maker concepts in Chapters 2–11 and the stories of technologists in Chapters 12–14 to make some recommendations about how to teach by making with a combination of 3D printing, maker electronics, and some old-fashioned tools, too. We also talk about how important it is to try things and fail. If you have to succeed all the time (as Joan saw at JPL), it limits the pace of learning. Low-cost making means you can have low-stakes failed projects, which is critical for learning engineering and science.

Making and the Common Core

If you are involved in education in the United States, you are probably very aware of the Common Core Initiative (www.corestandards.org). These new standards incorporate problem-solving and critical-thinking skills as central requirements for how students learn. We do not explore those links explicitly in this book, but note them here as something to explore further in the many resources available to teachers about the Common Core. If you search on the name of a technology and "Common Core," you will find a lot of aligned materials.

Educational Implications

Over the last several decades, manufacturing in the United States has gradually declined (although there is a lot of recent effort to change that). Because this means there are fewer jobs in manufacturing, traditional shop class has been languishing at many schools. If you couple declining interest with the liability issues of machine tools, you can see why schools with budget problems have been shuttering their shop classes. See, for instance, this article in *Forbes* by Tara Tiger Brown (www.forbes.com/sites/tarabrown/2012/05/30/the-death-of-shop-class-and-americas-high-skilled-workforce).

Creating a makerspace is a way to walk back into offering some sort of hands-on class, even if it is not a full-on shop class. The very things that make it hard to get started—that you do need to learn about the physical world—mean that students who have learned this way have a leg up over those who have never actually put anything together.

■ **Note** If you want a book to accompany design-focused learning in your classroom, Henry Petroski's books about the process of engineering design, most notably *To Engineer Is Human* (Vintage, 1992), focus on the role of trial, error, and failure in good engineering practice. Chapter 15 talks about this subject in depth. Donald Norman's design books, such as *The Design of Everyday Things* (Basic Books, 1988), are classics about how to observe the world and invent to meet real needs.

Broader Social Implications

The other impact of maker technologies (particularly 3D printing) has been to vastly lower the cost of making a prototype. Reducing the cost and raising the accessibility of a technology essentially democratizes it. This means that it is now possible for a seventh grader to create a prototype of a physical object that a few years ago would have required a professional modelmaker. 3D printing is the physical equivalent of low-cost computer graphic tools. This is nice for the seventh grader and may get her an A, but it is transformative for many professions, like product design. It also changes how those professionals work. If you are in the business of training people to be product designers, or engineers, or entrepreneurs for that matter, it is important that students learn how they will later work.

Making Prototyping Cheaper

If prototypes are cheaper and faster to make, the design process itself also becomes more iterative and more tolerant of failure along the way. Manufacturing may experience a broader sea change soon. A piece in *Harvard Business Review*'s blog by Peter Acton speculated that the ability to manufacture in small lots at home could create sweeping social change, as mass manufacturing starts to lose both its appeal and its price edge. (https://hbr.org/2014/12/is-the-era-of-mass-manufacturing-coming-to-an-end). These shifts imply that jobs will be shifting too—which means students need to be prepared differently for this emerging economic model.

Intellectual Property Issues

One of the biggest issues for these technologies is that making copies of physical objects becomes very easy. Intellectual property law will take a while to catch up with 3D printing in particular. How do we think about sharing (or selling) files that are then used to print physical things? When scanning technology gets more readily available, what will the rules be for copying something for your own use, or to sell it, particularly if you then build on the design and change it a lot? If you search on "intellectual property 3D printing," you can see the various discussions out there on these topics.

Summary

In this chapter, we introduced the concept of a maker and told you how the two authors came to be writing this book from their respective points of view (educator/engineer and hacker). We introduced the different technologies that are explained in depth in later chapters and pointed out that each chapter talks about the technology as well as summarizes what it can be used for, what you need to learn to use it, and how much it costs to get started. We also introduced the educational and broader social implications of these technologies.

CHAPTER 2

Arduino, Raspberry Pi, and Programming Physical Things

In Chapter 1, we discussed the different ways to think about using hands-on making to learn various subjects. In this chapter we introduce the basic nuts and bolts of commonly used open source electronics (microprocessors, single-board computers, and other components) and suggest paths to get started making things with these technologies.

Some of the components in this chapter (Arduino and Raspberry Pi) can and have been used for extremely sophisticated projects, up to and including an Arduino-controlled small spacecraft (the *Ardusat*, `www.ardusat.com`). We cover slightly less-ambitious applications of Arduino-class microcontrollers in Chapters 4, 6, 7 and 8. An *Arduino* is a microcontroller, which means it is not a whole computer. It is not intended to run multiple programs, drive a screen, and so on. It is meant just to control or monitor one or more devices. *Raspberry Pi* boards, on the other hand, really are full-blown computers and can run (moderate-sized) programs, handle a keyboard, and do many other things.

The big challenge of learning to use these technologies is that you have to learn several things at once. In most cases, you will need to write computer code in an appropriate integrated development environment (IDE) and be able to wire physical circuits, not to mention try to figure out what your system should actually do (Figure 2-1). If you are learning all that at the same time, it can be hard to figure out what is wrong if things do not work.

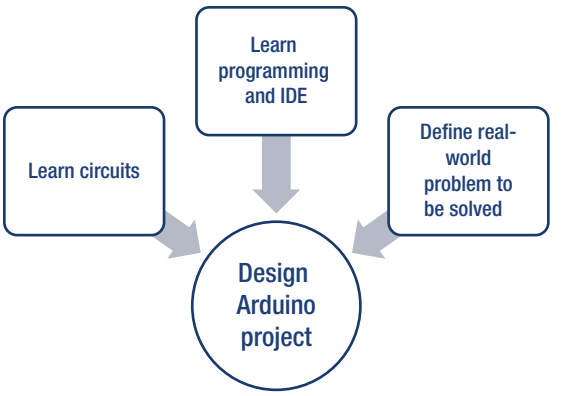


Figure 2-1. *Learning Arduino*

This chapter is largely Rich's world, but Joan has jumped in quite a bit to try to make things simpler. Some of this chapter might be a little more detailed than you want at this stage; in a few places, we've given you permission to skip to the next section if you want to. Since this chapter was a very tight blending of both our points of view, we just say "we" in this chapter rather than jump back and forth between our points of view. We are sure you will see a bit of both hacker and educator in the coming sections.

This field has a lot of terminology, and it is difficult to unravel it in an orderly way. So, we may use a word before we define it in depth. When we do that, we will give you a pointer to later in the chapter, so you can hold the term in your mind and see it discussed just a little farther on.

We have tried to minimize jargon as much as possible, but you will see a lot of the words here when you go to buy supplies, enroll in a class, or buy more detailed books, and we did not want to underprepare you, either. At the end of the chapter, we summarize what you need to learn for the different paths through this space and discuss what different key components typically cost so that you can budget as you get started. This is not so much a how-to chapter (we reference other books for that) as it is a why-would-you chapter. We want you to understand what you might do with a Raspberry Pi or Arduino and develop a basis of understanding that will carry you through the later, more specific application-focused chapters.

Some of the other electronics we talk about primarily in Chapter 8 (Circuit Stickers, LightUp) are intended to be used in more traditional classroom or home settings. They are geared toward simpler aspirations such as being able to design a basic circuit safely and as a gateway to more difficult programming.

One way to avoid having to deal with everything at once is to first learn how to write programs in a computer language called (confusingly) Processing. *Processing* is a programming language and development environment that is similar to those used by Arduinos. We introduce that here first as an option.

More typically, though, people learn how to use the technologies in this chapter by creating progressively more complex projects and thus learn incrementally more about each of the aspects. If you have some guidance, this is fine, but if not it can be difficult to know the difference between a "hard" and "easy" project before you embark. Figure 2-2 shows how we walk you through the possible learning paths. First we talk about Processing; then we backtrack a bit to give you some material about circuits so the rest is comprehensible; then finally we move on to the Arduino ecosystem itself. In later chapters we talk more about where else you can go with all this.

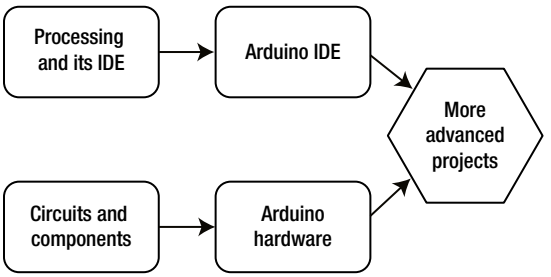


Figure 2-2. *The paths we follow in this book*

Processing and Arduino

One way to avoid trying to go in too many directions at once is to learn some basic commands in the computer language called Processing and then learn how to program an Arduino. Programming an Arduino is very similar to programming in Processing. Processing lets you use computer code to draw things on a screen in an animated way, so it could let students program an animation of a moving lever, for example. Arduinos allow you to control physical objects such as making a simple actual machine move.

Learning Processing

The Processing computer language is a simple language that is built on Java and is a lot like the C programming language. It is very good for creating animations. To test it out, go to the Processing tutorials page at https://processing.org/tutorials/overview/ and download the Processing IDE.

An IDE is software that allows you to develop and run code. The Processing IDE has a simple interface for writing and running code in the Processing language, and there is a reference for the IDE's interface (https://processing.org/reference/environment/). Once the IDE is installed, paste in some of the examples. Doing so will get you comfortable programming simple animations. If you then want to take the next step up and incorporate physical hardware into your simulations, you would then move to the Arduino IDE, covered in the next section, which is based on Processing's and is laid out very similarly.

Arduino and Its Ecosystem

Arduino is an open source platform comprised of a family of microcontroller boards and an IDE used for programming them. A microcontroller is an integrated circuit (IC), or computer chip, that includes a processor core, programmable input/output (I/O), and memory used to store the programs and data being used by the processor. To develop a program to run on an Arduino, you install the IDE on a laptop or desktop computer (a Mac, Windows, or Linux machine). Once you have written the code for the Arduino, you send it to the board via USB.

You can download the IDE from the Arduino main page (http://Arduino.cc). Figure 2-3 shows an Arduino and a Raspberry Pi next to each other for scale. The Arduino has been screwed down onto some acrylic next to a *breadboard*, which we talk about later.

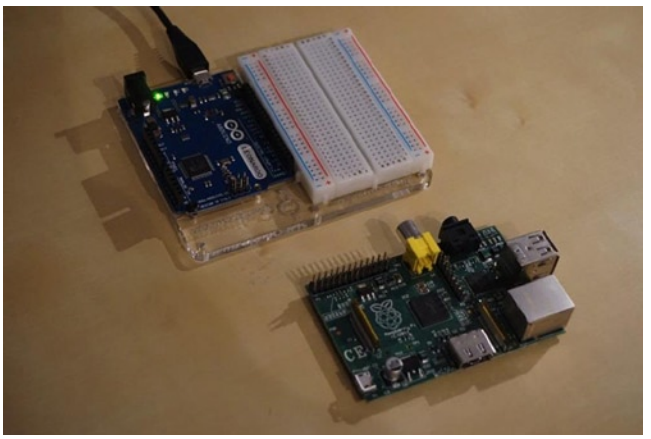


Figure 2-3. *An Arduino (top) mounted with a breadboard, and a Raspberry Pi*

An Arduino is essentially a computer built into a single chip that—although much less powerful than a desktop computer, or in most cases even a modern cellphone—is much more powerful than room-sized computers were in the 1970s. The Arduino's IDE is a text editor used for writing programming code to instruct the microcontroller on what to do, with built-in functions for compiling code written by the user into a series of simple instructions that the microcontroller will understand, and for communicating with the microcontroller to copy those instructions to its internal flash memory. An Arduino lets you interact with the physical world in some way that you program. You can program it to read data from a light sensor and start up a light-emitting diode (LED) if it gets dark, for example.

There are a variety of boards available with different shapes, sizes, and capabilities. Some of these are officially endorsed by the Arduino team and carry the Arduino name, though many are based to varying degrees on the open source designs that have been published for one of the official Arduino boards, or on one of the other unofficial boards. These unofficial boards are allowed to use the Arduino designs, but not the Arduino name except to say that they are *Arduino-compatible*. Many have names that include either the *Ardu-* prefix or the *-duino* suffix as shorthand for Arduino-compatible.

Official Arduino boards come in a range of prices starting around $20 (costs in this book are in U.S. dollars), whereas clones start at around $10 for the basic versions. Some counterfeit Arduino boards (ones that use both the open source designs that they are allowed to and the trademarked names and appearance that they are not) can be found for as little as $2.50. Although they usually work, they are likely to include inferior components, and our community discourages people from buying them for this reason—and because counterfeiting is an abuse of the spirit of open source (see Chapter 9).

■ **Tip** There are many good books and websites about Arduinos. However, it is important to find some beginner projects first. Just as you would probably not start teaching someone to drive at the Indy 500, it is best to learn these environments with relatively simple projects and work your way up. *Beginning Arduino* by Michael McRoberts, 2nd Edition (Apress, 2013) starts with a project to blink one LED and goes up from there. The Instructables website (`www.instructables.com`) has many different DIY projects; if you search on "beginner Arduino" or "beginner Raspberry Pi" on the site, you will find appropriate projects (although the definition of "beginner" on that site might be a little aggressive in some cases).

Arduinos and Arduino-compatible boards may not have as much computing power as a typical personal computer, but they can do some things a laptop computer cannot. If you want to connect to any electronic devices that are too simple to have a USB port or equivalent interface, your Mac or Windows computer probably will not be able to do it without something like an Arduino that is programmed to translate that data for it. Arduinos are commonly used for a variety of programmable electronics projects, including robots (Chapter 4), drones and other autonomous or semi-autonomous vehicles, 3D printing and other computer-controlled manufacturing (Chapter 3), translating and relaying or logging data from sensors (Chapter 6), wearable electronics (Chapter 7), playful computer interfaces (Chapter 8), and many other tasks that involve linking simple electrical interfaces either with their limited computing power or translating that data into the more sophisticated protocols required by a more powerful computer. Arduinos are also smaller and cheaper than a typical laptop or desktop, and use less power, which makes them better for embedding in projects. (See the next section for more detail.)

An Arduino, unlike a Mac or Windows computer, does not run an operating system or multitask between multiple programs. It has a single program that begins running when it powers up, after a brief pause for the bootloader to check whether you are trying to upload a new program, and continues until the power is switched off. This makes it better for certain operations where precise timing is critical, such as sending commands to move stepper motors. (A stepper motor is a motor which turns a shaft in discrete, precise increments, often used for robotics projects, as opposed to a brushed motor that just turns continuously when a voltage is applied; see the section on stepper motors later in this chapter.) Computer-numerically-controlled (CNC) machine tools like mills and lathes, which use stepper motors to move their tools, are typically built with a computer controller, but using an Arduino-equivalent to control them might work better for precise timing. Open source 3D printers are typically built this way.

All the current Arduino boards use chips made by Atmel Corporation, but there are other similar products. As of this writing, most Arduino boards and their clones use 8-bit microcontrollers from Atmel's AVR family of products, but Microchip Technology's PIC family of products has similar capabilities, and Parallax Inc.'s BASIC Stamp is also comparable. Religious wars are fought on the Internet over which is best,

and each product line (as well as each individual product) has its own strengths and weaknesses compared to the rest of the ecosystem. It is our opinion that the ubiquity and beginner-friendliness of the Arduino environment makes it a clear winner, even when the others might be more powerful. Although the others might be better for specific uses, the wealth of free information and shared projects make the Arduino much more useful for those just getting started.

Interfacing an Arduino with the Real World

The power of an Arduino is that it can interact with the physical world. This section talks in some detail about how that works. Figure 2-4 illustrates the general idea.

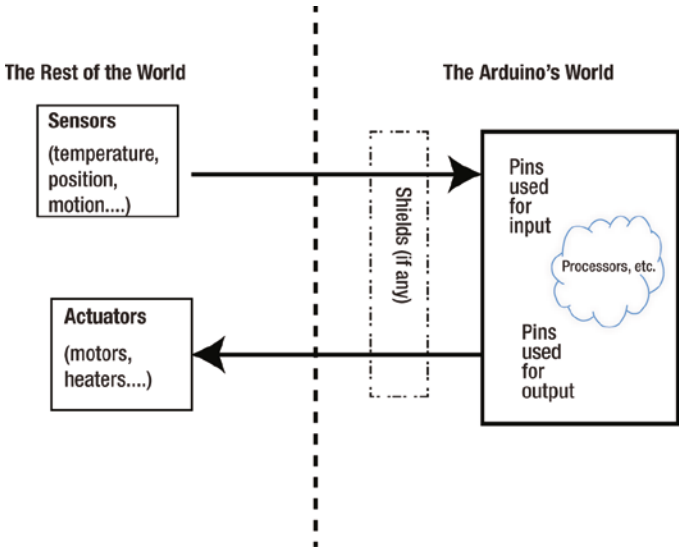


Figure 2-4. *How an Arduino interacts with the physical world*

Pins on a computer chip are electrical contacts facing down toward the circuit board that are either soldered directly to the board or connected to a socket that is. Some pins are used to provide power to the chip, but most are used to either take a signal in from something (*input pins*) or send a signal to something (*output pins*). Arduinos process the signals coming in and then send out signals based on what they perceive. This part is a little technical, but if you are doing a first pass through now, we suggest you come back to this section later, because some of the insights here will be important in later chapters that talk about how to use Arduinos and associated technologies in bigger projects. We give you our permission to skip ahead to the section "Circuit Design and Components" if you feel like you do not need to know details right now.

Arduinos have General Purpose Input/Output (GPIO) pins that can be configured either as inputs that can read things like analog voltages and pulse frequencies that various sensors produce, or as outputs that can trigger LEDs, motors, small display modules, and so on. Any of the GPIO pins on an Arduino can be configured for use as either a digital input or a digital output. You can change which mode is being used at any time. Each program generally sets each pin's mode at the beginning and does not change it, though there are exceptions.

People typically think of digital hardware as being binary systems that have just two values (think *on* and *off*). Depending on context, these values might be *true* and *false,* or 1 and 0, or HIGH and LOW. The Arduino has some tricky ways of finessing this to get other values in between to emulate a