Table of Contents

Meta-Data

Lesson Goals

- Students will understand the 15 design principles described in the lesson.
- Students will understand the four sets of design principles that gave rise to the 15 principles covered here.

Lesson Outcomes

- Students will be able to identify examples of violations of these 15 design principles in interfaces.
- Students will be able to evaluate interfaces according to these principles.
- Students will be to design interfaces that follow these design principles.

Assessments

- Students will reflect on the application of the lesson's concepts to their chosen area of HCI.
- Students will engage in a short design task based on the lesson's concepts.
- Students will complete a short answer assignment in which they critique a provided interface from the perspective of the lesson's concepts.
- Students will complete a short answer assignment in which they select an interface to critique from the perspective of the lesson's concepts.
- Students will complete a short answer assignment in which they design a revision of one of the critiqued interfaces from the perspective of the lesson's concepts.

Lesson Plan

- Students will begin by learning about the four sets of design principles covered in the lesson.
- Students will then learn about each lesson one by one.

Script

2.5.1 Introduction

2.5.1.1 Headshot Studio

- [C] David talking
- [A] Clips of the lesson on the right
- Over the main years of HCI development, experts have come up with a wide variety of principles and heuristics for designing good interfaces.
- None of these are hard rules like the law of gravity or something, but they're useful guidelines to keep in mind when designing interfaces.
- [B] Topic; Don Norman's Six Principles
- Likely the most popular and influential of these are Don **Norman**'s six principles of design.
- [B] Topic; Constantine's and Lockwood's Six Principles
- Larry **Constantine** and Lucy Lockwood have a similar set of six principles of user interface design, with some overlap but also some distinction.
- [B] Topic; Nielsen's Ten Heuristics

- Jakob **Nielsen** has a set of ten heuristics for user interface design that can be used for both design and evaluation.
- [B] Topic; Seven Principles of Universal Design
- And while those are interested in general usability, there also exists a set of seven principles called principles of **universal design**.
- These are similarly concerned with usability, but more specifically for the greatest number of people.
- Putting these sets together, we'll talk about fifteen unique principles of interaction design.

2.5.2 The Sets

2.5.2.1 Tablet Studio

- [V] Icons for four sets across the top -- logos, book covers, etc.: Norman's book "Design of Everyday Things", Nielsen's book "Usability Engineering", Constantine & Lockwood's "Software for Use: A Practical Guide to the Essential Models and Methods of Usage-Centered Design", and the logo for the Center for Universal Design at NC State
- We'll talk about four sets of design principles.
- [V] Norman's book enlarges to the left, text appearing on the right.
- Donald Norman outlines seven principles in his seminal book Design of Everyday Things. These are likely the most well-known set of principles, covering general usability.
- [V] Norman's book recedes, Nielsen's book takes its place
- Jakob Nielsen has a list of ten usability heuristics to guide design. Many of Norman's principles are similar to some principles from Nielsen's list as well.
- [V] Constantine and Lockwood's book comes up
- Larry Constantine and Lucy Lockwood's book "Software for Use" covers usage-centered design with a set of seven principles.
- [V] UD logo comes up
- And the Center for Universal Design at NC State University has posed its own list of seven principles for universal interface design.
- [V] Logo recedes, logos disappear, replaced by icons
- To make this lesson a little easier to follow, I've tried to merge these four sets of principles into one larger set, capturing the overlap between many of them.
- We'll go through these fifteen principles in this lesson.
- These principles are intended to distill out some of the overlap between these four prominent sets of design principles.
- [V] Icons recede, show table

- This table shows those fifteen principles, my names for them, and which sets they come from.
- Note that my fifteen principles are just an abstraction or summary of these sets of principles -- you should make sure to understand these sets as well.
- And note also that these aren't the only sets out there. At the end of the lesson, we'll chat about a few more, and we'll also mention when others apply within this lesson as well.

2.5.3 Discoverability

2.5.3.1 Tablet Studio

- [V] Three definitions from three different sets (will find)
- Don Norman's principle of discoverability is very similar to the Visibility Principle from Constantine and Lockwood, as well as Nielsen's idea that Recognition is preferable to Recall.
- The idea behind all three of these principles is that relevant functions should be made visible so that the user can discover them, as opposed to having to read about them in the documentation or learn them through a tutorial.
- Let's take an example real quick.
- <exit presentation, revealing PowerPoint>
- Here in PowerPoint, there are a number of menus available at the top, as well as some toolbars.
- The effect here is that I can browse the different functions available to me.
- I can discover what's there.
- For Nielsen, this means that I don't have to remember all of them: I can simply recognize them in these menus and toolbars.
- This is true at the application level, but it's often not true at the operating system level.
- For example, on a Mac, I can use Cmd+Shft+4 to start taking a screenshot of an area of my screen.
- <do so>
- However, the only way I know of to find that is to Google it or read it in a manual -- it isn't discoverable or visible on its own, and you might never even realize it's possible.
- So, the principle of Discoverability advocates that functions be visible to the user so they can discover them, rather than relying on them learning about them elsewhere.
- Constantine and Lockwood's principle of Visibility would add on to this that we should get too crazy.
- [V] Window with way too many toolbars

- We want to make functions discoverable, but that doesn't mean just throwing everything on the screen.
- We want to walk a line between discoverability and simplicity.

2.5.4 Design Challenge: Discoverability and Gestures

2.5.4.1 Headshot Studio (Morgan)

- [C] Morgan sitting at the table
- Discoverability is one of the challenges for designing gesture-based interfaces.
- To understand this, let's watch Morgan do some ordinary interactions with her phone.
- <Morgan gets a loud phone call and fumbles trying to reject it>
- <Morgan holds up the phone and takes a screenshot>
- <Morgan takes a selfie by pressing the button, struggling a bit>
- <Morgan dials a number, then presses the dial button>
- [B] Reject a call
- [B] Take a screenshot
- [B] Take a selfie
- [B] Make a phone call
- We just saw Morgan do four things with the phone: **reject** a call, **take** a screenshot, **take** a selfie, and **make** a phone call.
- For each of those, this phone actually has a corresponding gesture that would have made it easier.
- She could have just turned the phone over to reject the call, or said "shoot" to take the selfie.
- The problem is that these are not discoverable.
- Having a menu of voice commands would kind of defeat the purpose of saving screen real estate and simplicity through gestures and voice commands.
- So, brainstorm a bit: how would you make gesture commands discoverable?

2.5.4.2 Exercise

- [E] Box for exercise entry

2.5.4.3 Headshot Studio (Morgan)

- [C] Morgan sitting at the table performing those actions
- There's a lot of ways we might do this, from training in advance to tutoring in context.

- For example, we might use the title bar of the phone to just briefly flash a message letting the user know when something they've done could have been triggered by a gesture or voice command.
- That way, we're delivering instruction in the context of the activity.
- We could also log those so a user could check back at their convenience and see tasks they've performed that could have been triggered in other ways.

2.5.5 Simplicity

2.5.5.1 Tablet Studio

- [V] Window with way too many toolbars
- There often exists a tension between discoverability and simplicity.
- On the one hand, discoverability means that you need to be able to find things.
- But how can you find things if they're not accessible or visible?
- That's how you get interfaces like this: way too many things visible, and ironically as a result, it becomes hard to actually find what you're looking for.
- [V] Three definitions from three different sets
- Simplicity is a part of three of our sets of principles: Nielsen's heuristics, Constantine and Lockwood's principles, and the Universal Design principles.
- <go through definitions>
- In some ways, these are about designing interfaces, but they cover other elements as well.
- [V] Old vs. new blue screen of death
- For example, Nielsen's heuristic can be applied to error messages: the user should only be given as much information as they need.
- [V] This article or image.
- Simplicity as a universal design principle is especially interested with whether people of different experiences, knowledges, or languages can figure out what to do.
- These two signs communicate the same information, but while the one on the left requires a lot of cognitive load and language skills, the one on the right can probably be understood with little effort and experience.

2.5.6 Affordances

2.5.6.1 David's House (Car)

- [C] David by his car

- One way to keep design simple and usable is to design interfaces that, by their very design, tell you how to use them.
- Don Norman describes these as affordances: the design of things affords, or hints at, the way they're used.
- This is also similar to the familiarity principle from Dix et al.
- This is extremely common in the physical world because the physical design of objects is connected to the physical function they serve.
- Buttons are meant to be pressed, handles are meant to be pulled, knobs are meant to be turned.
- You can simply look at it and understand how you're supposed to use it.

2.5.6.2 Tablet Studio

- [V] Definition of Affordance.
- On affordances, Don Norman writes <read definition>.
- He goes on to say <read second part>.
- Note the importance of the user in this. Our affordances are defined by who the user is.
- [T] The desktop -- I'll find examples of these in the OS to use
- The challenge is that in the virtual, computer world, there are no such inherent connections between the design and function of an interface.
- For example, when I click a button on an interface, there is a visualization of the button being depressed, but it didn't have to look that way.
- It could have popped out, it could have disappeared, it could have changed colors.
- But the design of this button, with the illusion of raised edges around the sides and the visualization of being depressed when it's clicked, hints at how it is meant to be used.
- When designing computer interfaces, we have to create that naturalness manually.
- We can do that in a number of ways.
- We could for example, visualize the space of options. A volume control does this: the vertical line shows us the list of options, the horizontal line shows the current location, and it's natural to drag the bar around in the space of options.
- We could leverage analogies to physical devices. For example, in a book-reading app, we might use swipes back and forth to turn pages because that mimics the affordances of a physical book.
- Of course, there are also actions in the virtual world that have no real-world analogy, such as pulling up a menu on a mobile site.
- [T] Mobile version of LucyLabs
- In that case, we may use Signifiers. Signifiers are a principle in Norman's more recent editions.

- Signifiers are in-context instructions, such as arrows to indicate which way to swipe or a menu icon to indicate how to access the options.
- In this way, we can kind of create our own affordances by creating an intuitive mapping between controls and their effects in the world, being consistent with what others have done in the past.

2.5.6A Affordances Vocabulary

2.5.6A.1 Tablet Studio

- [V] Affordance, perceived affordance, signifier, and two doors
- It's important to note that the language with which we talk about affordances is famously somewhat imprecise.
- Norman's technical definitions of affordances are a little different than what we used here.
- Affordances, to Norman, are actually inherent properties of a device.
- For example, a door bar like this one has the inherent property that the handle moves into the cross bar and opens the latch: that is the affordance.
- A perceived affordance is a property attributed to the object by a human observer.
- This can be a subtle difference: here, the perceived affordance would be pushability. To 'push' is a human behavior, so pushability must be a perceived affordance because it relies on someone to do the pushing.
- A perceived affordance can be inaccurate: Norman famously complains about doors that have a handle, which gives the perception of pulling, on doors that are to be pushed. That's a place where a perceived affordance and an actual affordance are in conflict.
- A signifier, then, is anything that helps the perceived affordance match the actual affordance.
- For example, the black bar on the push door tells the user where to push. It signifies the affordance.
- A sign that says 'push' on a door handle would also be a signifier to try to alleviate the conflict between the perceived affordance and the actual affordance.
- For this reason, we can't add affordances. Affordances are inherent in our systems.
- Instead, we can add signifiers, and signifiers raise the likelihood that the user will accurately perceive the affordances that are already present.
- With the technical definitions, saying "I added an affordance to the interface" is like saying "I added tastiness to that dish" or "I added beauty to that painting". Affordances, tastiness, and beauty are things that arise as a result of adding signifiers, oregano, or a pretty shade of blue.

- In practice, however, these distinctions around this vocabulary are often disobeyed.
- It's not uncommon to hear people saying, "We need an affordance so users know they can pull that handle" or something similar.
- And in my mind, there's no harm in that: the distinctions between these terms are useful when developing the theory of HCI, but in day-to-day design, we usually know what we're talking about when we "misuse" these terms.

2.5.7 Mapping

2.5.7.1 Tablet Studio

- [V] Definitions
- Norman and Nielsen both talk about the need for a mapping between interfaces and their effects in the world.
- <Norman's definition>
- For example, these book icons help you map these quotes to the books from which they were taken.
- <Nielsen's definition>
- Note that these two principles are subtly different, but strongly related. Nielsen's heuristic describes the general goal, while Norman's principle describes one way to achieve it. Strong mappings help make information appear in a natural and logical order.
- [V] Monitor display
- When setting the arrangement of different monitors, the visual display creates a clear mapping to their physical arrangement.
- If instead this was just shown as a list of pixel locations, it would be much more difficult to map the visualization to its meaning.
- Mappings and Affordances are similar principles, but with a clear difference.
- [T] Volume control image, or bring up the volume control directly
- Affordances were about creating interfaces where their design suggested how they are used.
- The presence of a dot in the middle of a line like this in many ways affords that it can be dragged back and forth along this axis.
- However, on its own, this doesn't tell me what the effect will be. It affords the interaction, but it might not map to the effect.
- The presence of these icons on either side, as well as the visualization of its relative position on the line, together create a mapping between the control and its effect.
- Mapping refers to creating interfaces where their design makes it clear what the effect will be of using them.

- With this volume control, the arrangement of the controls makes it clear what to do, and the visualization underneath makes it clear what will happen when I do it.

2.5.8 Mapping and Switches

2.5.8.1 David's House (Kitchen)

- A good example of the difference between affordances and mapping is a lightswitch.
- A lightswitch very clearly affords how you're supposed to use it.
- You're supposed to flip it.
- But these switches have no mapping to what will happen when I switch them.
- I can look at it and clearly see what I'm supposed to do, but I can't tell what the effects is going to be in the world.
- <David walks over to the stove>
- Contrast this with the dials on my stove.
- There are four dials.
- But each is augmented with this icon that indicates which burner is controlled by which dial.
- Thus, there is a mapping between the controls and the effects.
- <David returns to the light switches>
- So, how might you redesign these light switches to create not only affordances, but also mappings?
- [C] Shots of each light as I run through them?
- If relevant, this one turns on the kitchen light, this one turns on the counter light, and this one turns on the breakfast room light.

2.5.8.2 Exercise

- [E] How might you modify this control to create a mapping between the switches and their effects?
- [E] (box for answer)

2.5.8.3 David's House (Kitchen)

- [C] David talking by the light switches
- There are a few things we could do.
- Maybe we put a small letter by each switch to indicate what they go to: K for kitchen, C for counter, B for breakfast room.

- Maybe we actually put icons on them demonstrating what kind of light it is. One for the bowl light in the kitchen, two small ones for the counter lights, and a chandelier for the one in the breakfast room.
- But likely the easiest thing is actually the way that it was designed.
- [C] Shot of David from across the room
- The lights from left to right correspond to the switches from left to right, so each light is controlled by the switch closest to it.
- That forms an intuitive mapping.

2.5.9 Perceptibility

2.5.9.1 Tablet Studio

- [V] Definitions
- Perceptibility refers to the user's ability to actually perceive the state of the system.
- <definitions>
- These definitions are actually pretty different -- Nielsen is interested in perceptibility of the state of the system, like whether things are on or off, while universal design is concerned with the perceptibility of information, like how easy signs can be read.
- Nonetheless, though, in designing interfaces we often end up addressing both of these things together: we want to make the state of the system easily perceptible.
- [V] Norman's feedback definition appears
- Norman's notion of feedback is also very similar to perceptibility.
- <definition>
- In practice, though, Norman's notion of feedback so fundamentally underlies several of these principles that it is difficult to categorize it under one umbrella.

2.5.9.2 David's House (Kitchen)

- [C] David by light switches
- Things like light switches and oven dials do this very nicely already.
- I can look at a light switch and determine whether the system it controls is on or off based on whether the switch is up or down.
- Same with the dial: I can immediately see where the dial is set.
- But there's a common household control that flagrantly violates the principle of perceptibility.

2.5.9.3 David's House (Living Room)

- [C] David under the ceiling fan

- Here's our ceiling fan. You might have one like it.
- It has two chains. One controls the light, one controls the fan speed, but only when the switch is on.
- First, the mapping here is awful: there's no indication which is which.
- But worse, the fan chain, which happens to be this one.
- <David pulls the wrong chain, light turns off, eyeroll, turns it back on>
- See what I mean?
- The fan chain...
- <David pulls the right chain>
- Doesn't give any indicator of which setting the fan is on currently.
- I don't honestly even know how many settings it has.
- I don't know if pulling it makes it go up and then down, up and then off, down and then off…
- Whenever I use it, I just pull it, wait ten seconds, see if I like the speed, and pull it again.
- And this is all only if the wall switch is on!
- Now of course, people have resolved this with dials, or other controls, and yet these dang chains still seem to be the most common approach, despite this challenge with perceptibility.

2.5.10 Consistency

2.5.10.1 Tablet Studio

- [V] Three definitions from three different sets
- Consistency is a principle in Norman's principles, Nielsen's heuristics, and Constantine and Lockwood's principles.
- <<run through the definitions>>
- This is also similar to Dix et al's idea of generalizability.
- The general idea across all of them is that we should be consistent both within and across interfaces to minimize the amount of learning the user needs to do to learn our interface.
- In this way, we kind of create affordances of our own: unlike traditional physical affordances, there is no physical reason for the interface to be designed that way, but by convention we create expectations for users.
- [T] Web pages of different examples opened in tabs
- One great example of this is the conventions surrounding links in text.
- For whatever reason, an early convention on the internet was for links to be blue and underlined.

- Now, when we want to indicate to users that some text is clickable, what do we do? Generally, we make it blue, and we underline it.
- Sometimes we change this: maybe we just make it a contrasting color, maybe we just underline it, but the most fundamental convention is still blue and underlining.
- Most of the interfaces we design will have a number of functions in common with other interfaces, so by leveraging the way things have been done in the past, we can help users understand our interfaces more quickly.
- [T] Back to PowerPoint, open a menu
- Other common examples of consistency in interface design would include things like using consistent hotkeys for copy, paste, and select all.
- Ordering the menus as File, Edit, View, etc.
- Putting options like Save and Open under file, etc.
- We don't even tend to think about these things when we're using an interface until we encounter one that defies our conventions: and yet, someone had to consciously decide to be consistent with established norms.
- And if you've ever used an interface that didn't adhere to these norms, you most likely found it quite jarring.

2.5.11 Consistency: The Curious Case of Ctrl+Y

2.5.11.1 Tablet Studio

- [V] Variety of interfaces with the Edit menus open, showing Ctrl+Y is usually 'redo'.
- One of my favorite examples of how consistency matters comes from Microsoft's IDE Visual Studio.
- And to be clear: I adore Visual Studio, so I'm not just piling onto it.
- As you see here, in most interfaces, Ctrl+Y is the 'redo' hotkey. If you hit undo one too many times, you can hit Ctrl+Y to 'redo' the last 'undone' action. On Mac, this is usually Cmd+Y.
- [V] Visual Studio comes up, edit window open
- In Visual Studio, by default it's… Shift+Alt+Backspace.
- What's worse: Ctrl+Y is the 'delete line' function.
- So, if you're pressing Ctrl+Z a couple times to look back at what you've changed, then press Ctrl+Y, it deletes the current line… thus making a new change, and blocking access to the original series of actions that you would have redone.
- It's infuriating.
- But yet, it isn't without its reasons.
- The reason: consistency.
- [V] WordStar fades in

- Ctrl+Y was the hotkey for delete line in WordStar, one of the very first word processors, before Ctrl+Y was the hotkey for the more general redo function.
- Y in this context stood for 'Yank', and Ctrl+Y had been used to delete a line for from WordStar through Visual Basic 6, the predecessor of Visual Studio.
- So, in designing Visual Studio, Microsoft had a choice: stick with the convention from Visual Basic 6, or stick with the convention from the rest of their software.
- [V] WordStar fades out
- They chose to be consistent with the previous versions, and they've stayed consistent with that ever since.
- So in trying to maintain the consistency principle, you'll encounter some challenges. There may be multiple conflicting things with which you want to be consistent. There may be questions about whether a the value of a change is worth the drop in consistency.
- [B] Elsewhere in HCI; Unit 3: Research Methods
- These are things to test with users, which we'll talk about in the next unit.

2.5.12 Flexibility

2.5.12.1 Tablet Studio

- [T] PowerPoint open
- Depending on your expertise with computers, there's a strong chance you've found yourself on one side or the other of the following exchange.
- One person is watching the other use a computer.
- The person using the computer repeatedly goes to Edit and selects Cut, edit and selects Paste, etc. to access those commands.
- The person watching insists that they can just use Ctrl+X and Ctrl+V.
- The person working doesn't understand why the person watching cares.
- The person watching doesn't understand why the person working won't use the more efficient method.
- In reality, they're both right.
- [V] Reenter presentation, two definitions of flexibility open
- These two options are available because of the principle of flexibility, from both Nielsen's heuristics and the principles for universal design
- <go through definitions>
- Dix et al. also have a category of principles called Flexibility principles that advocate user customizability and supporting multiple designs for the same task.

- Nielsen is more interested in catering to both novice and expert users, while the principles of universal design are more interested in accommodating users of various abilities and preferences, but the underlying principle is the same.
- Wherever possible, we should support the the different interactions in which people engage naturally, rather than forcing them into one against their expertise or preference.

2.5.13 Equity

2.5.13.1 Tablet Studio

- [V] Definitions of principles of equity and flexibility side-by-side
- The principle of flexibility in some ways clashes with the principle of equity, but both come from the principles of universal design.
- In reality, though, they're actually complementary of one another.
- Equity is largely about making the user experience the same for all users, and flexibility is a means to achieve that.
- User experience in this instance means treating every user like they're within the target audience and extending the same benefits to all users, including things related to privacy and security.
- [T] Georgia Tech password reset requirements come up
- Password reset requirements could be seen as one example of equity in action.
- We want to design for both expert and novice users.
- Expert users understand the value of a complex password, but novices might not.
- So, we want to extend the same security provisions to novices that we extend to experts, and we do that by requiring complex passwords from all users.
- In the process, we might frustrate novice users a little bit, and you could see this as a violation of flexibility, but the important thing here is we're extending the same benefits to everyone.
- That's equitable treatment.

2.5.14 Ease & Comfort

2.5.14.1 Tablet Studio

- [V] Definitions of principles of ease and comfort
- The principles of universal design have two additional principles that relate to equitable treatment, specifically in terms of physical and interaction.
- <<definitions>>

- In the past, these principles did not have an enormous amount of application to HCI.
- They're related more to the design of physical environments rather than technological ones.
- But as more and more interfaces are becoming equipped with computers, we'll find HCI dealing with these issues more and more.
- [V] Mobile interface with buttons
- For example, when deciding on the size of buttons on a mobile interface, we should take into consideration that some users may have tremors that make it more difficult to interact precisely.
- As we get into areas like wearable computing and virtual reality, these issues of ease and comfort will become more and more pertinent.

2.5.15 Structure

2.5.15.1 Tablet Studio

- [V] Definition of structure principle
- The structure principle is concerned with the overall architecture of a user interface.
- In many ways, it's more closely related to user interface design than HCI more generally.
- What's interesting to me about the structure principle is that it borrows from a form of user interface design that predates computers considerably.
- We find many of the principles we learned in designing newspapers and textbooks apply nicely to user interfaces as well.
- [V] An image of the Wall Street Journal
- For example, this is the Wall Street journal print edition from several years ago
- [T] Show WSJ.com
- And here's the Wall Street Journal web site.
- Notice many of the structural principles present in the print version are present here as well.
- Lines and spacing set apart different categories of articles.
- Headlines are in bolder font while text is in smaller.
- There are differences, of course: web sites can link to articles while physical papers cannot.
- However, we see lots of the same principles at work here that were at work in the physical layout.
- Those are largely parts of structure: organizing things in an intuitive way that groups similar parts, separates dissimilar ones, and helps the user navigate what they're consuming.

2.5.16 Constraints

2.5.16.1 Tablet Studio

- [V] Definitions
- In designing user interfaces, our goal is typically to make the interface usable.
- A big part of usability is accounting for user error.
- Many design theorists argue that there is no such thing as user error: if the user commits an error, it was because the system was not structured in a way to prevent or recover from it.
- One way we can avoid error is by preventing the user from performing erroneously in the first place.
- This is the idea of constraints: constraining the user from only performing the correct actions in the first place.
- <<definitions>>
- Both these approaches refer to the need to stop faulty user input before it is received.
- [T] Go to a well-designed credit card screen
- This is a principle you may already encounter a lot.
- Take this credit card form, for example.
- There are a lot of ways I could mess this up.
- I could accidentally type a letter, or type the entire year.
- [V] Return to slide
- In its simplest form, constraints can be described as: prevent the user from entering input that wouldn't have worked anyway.
- [V] Hide Nielsen's definition
- Norman takes this a step further, though.
- Norman breaks constraints down into four sub-categories.
- These aren't just about preventing wrong input; they're about ensuring correct input.
- They're about making sure the user knows what to do next.
- [V] Show physical
- Physical constraints are those that literally, physically prevent you from performing the wrong action.
- A three-prong plug, for example, can only physically be inserted one way, preventing mistakes.
- USBs can only be inserted one way as well, but that constraint doesn't arise until you try to do it incorrectly, which isn't as optimal.
- [V] Cultural appears

- Cultural constraints are those rules that are generally followed by different societies, like facing forward on an escalators or forming a line while waiting.
- In designing, we might be able to rely on these, but we should be cautious of intercultural differences.
- [V] Semantic appears
- Semantic constraints are those that are inherent to the meaning of the situation. They're similar to affordances in that regard.
- For example, the purpose of a rearview mirror is to see behind you, so therefore mirrors must reflect from behind.
- In the future, the meaning might change: autonomous vehicles might not need mirrors for the passengers, so the semantic constraints of today may be gone tomorrow.
- [V] Logical appears
- Finally, the last kind of constraint is the logical constraint.
- Logical constraints are things that are self-evident based on the situation.
- For example, imagine building some furniture. When you reach the end, there is only one hole left, and only one screw. Logically, the screw is constrained to go in the one remaining hole.

2.5.17 Reflections: Constraints

2.5.17.1 David's House (Car)

- [C] David sitting in the open driver's seat
- A lot of the principles we talk about are cases where you might never even notice if they've been done well.
- They're principles of invisible design where succeeding allows the user to focus on the underlying task.
- Constraints are different, though, in that they actively stand in the user's way: they become more visible.
- That's often a bad thing, but in the case of constraints, it serves the greater good.
- Constraints might prevent users from entering invalid input, or force users to adopt certain safeguards.
- So, of all the principles we've discussed, this might be the one you've noticed.
- Can you think of any times you've encountered interfaces with constraints?

2.5.17.2 Exercise

- [E] What is a time when you encountered an interface that constrained your interaction with it?
- [E] (box for input)

2.5.17.3 David's House (Car)

- [C] David sitting in the open driver's seat
- I have a kind of interesting example.
- I can't demonstrate it well because the car has to be in motion.
- But on my Leaf, there's an options screen.
- It lets you change the time, date, and other options on the car.
- And you can use it until the car starts moving.
- At that point, the menu blocks you from using it, saying you can only use it when the car is at rest.
- That's for safety reasons. They don't want people fiddling with the options while driving.
- What makes this interesting, though, is that it's a constraint that isn't in service of usability.
- It's in service of safety.
- The car is made less usable to make it more safe.

2.5.18 Tolerance, Feedback, & Documentation

2.5.18.1 Tablet Studio

- [V] Definitions of tolerance
- We can't constrain away all errors all the time, though.
- So, there are two principles for how we deal with errors that do occur: feedback and tolerance.
- Tolerance means that users shouldn't be at risk of causing too much trouble accidentally.
- <<definitions>>
- Dix et al. also refers to this as recoverability.
- Nielsen's definition is most interested in supporting user experimentation; the system should tolerate users poking around with things.
- Constantine and Lockwood's definition and the principle for universal design are a little closer to traditional mistakes.
- [V] Raskin's principle comes up
- Jef Raskin poses this as a more humorous law of interface design: "A computer shall not harm your work or, through inactivity, allow your work to come to harm."
- So, we first have to make sure that the system prevents the user from doing too much damage accidentally.
- [V] Definitions of feedback

- Second, the system should give plenty of feedback so that the user can understand why the error happened and how to avoid it in the future.
- <<definitions>>
- Notice that Norman, Constantine, and Lockwood are all interested not just in feedback in error detection, but also in feedback in the interface more generally.
- [V] Feedback cycle comes up
- [B] Elsewhere in HCI: Lesson 2.2, Feedback Cycles
- That's why we spend so much time talking about **feedback** cycles: it's arguably the most fundamental principle.
- Dix et al. have it in their principles as well, as part of observability and responsiveness.
- But since we talk about it on its own so much, here let's emphasize Nielsen's version, error recognition.
- [V] Definition of documentation
- Finally, Nielsen has one last heuristic regarding user error: documentation.
- One goal of usable design is to avoid the need for documentation altogether.
- We want users to just interact naturally with our interfaces.
- However, Nielsen advocates last that we supply comprehensive documentation anyway, just in case.

2.5.19 Exploring HCI: Design Principles and Heuristics

2.5.19.1 Headshot Studio

- [C] David talking
- We've talked about a bunch of different design principles in this lesson.
- How these design principles apply to your design tasks will differ significantly based on what area you're working in.
- In gestural interfaces, for example, constraints present a big challenge because we can't physically constrain a user's movement.
- We have to give them feedback or feedforward in different ways.
- If we're working in particularly complex domains, we have to think hard about what simplicity means: if the underlying task is complex, how simple can and should the interface actually be?
- We might find ourselves in domains with enormous concerns regarding universal design: if you create something that a person with a disability can't use, you risk big problems, both ethically and legally.
- So, take a few moments and reflect on how these design principles apply to the area of HCI you've chosen to investigate.

2.5.20 Other Sets of Principles

2.5.20 Tablet Studio

- [V] Original 15 principles
- So, I've attempted to distill the 29 combined principles from Don Norman, Jakob Nielsen, Larry Constantine, Lucy Lockwood, and the Centre for Universal Design into these 15.
- [V] Table of principles
- I do recommend reading the original lists as well to pick up on some of the more subtle differences in the principles I've grouped together, especially perceptibility, tolerance, and feedback.
- [B] Elsewhere in HCI: 2.6 Mental Models & Representations
- Note also that in more recent editions, Don Norman has one more principal: conceptual models. That's actually the **subject** of an entire lesson in this class.
- [V] List of other lists coming up as they're described
- These also aren't the only lists of principles.
- Dix, Finlay, Abowd, and Beale propose three categories of principles: Learnability for how easily a new user can grasp an interface; Flexibility for how many ways an interface can be used; and Robustness for how well an interface gives feedback and recovers from errors. We talk about their learnability principles when we discuss mental models.
- Jill Gerhardt-Powals has a list of principles for cognitive engineering aimed especially at reducing cognitive load. Her list has some particularly useful applications data processing and visualization.
- In The Human Interface, Jef Raskin outlines some additional revolutionary design rules. I wouldn't advocate following them necessarily, but they're interesting to see a very different approach to design.
- In Computer Graphics: Principles and Practice, Jim Foley and others give some principles that apply specifically to 2D and 3D computer graphics.
- And finally, Susan Weinschenk and Dean Barker have a set of guidelines that provide an even more holistic view of interface design, including things like linguistic and cultural sensitivity, tempo and pace, and domain clarity.

2.5.21 Conclusion

2.5.21.1 Headshot Studio

- [C] David talking
- [A] Clips of the lesson on the right
- In this lesson, I've tried to take the various different lists of usability guidelines from different sources and distill them down to a list you can work with.
- [B] Topic; Don Norman's Six Principles
- [B] Topic; Jakob Nielsen's Ten Heuristics
- [B] Topic; Constantine's and Lockwood's Six Principles
- [B] Topic; Universal Design's Seven Guidelines
- We combined the lists from **Don** Norman, **Jakob** Nielsen, **Larry** Constantine, Lucy Lockwood, and the **Institute** for Universal Design into fifteen principles.
- Remember, though, these are just guidelines, principles, and heuristics: none of them are unbreakable rules.
- You'll often find yourself wrestling with tensions between multiple principles.
- There will be something cool you'll want to do, but only your most expert users will understand it.
- Or, there will be some new interaction method that you want to test, but you aren't sure how to make it visible or learnable to the user.
- These principles are things you should think about when designing, but ultimately they'll only get you so far.
- You need needfinding, prototyping, and evaluation to find what works in reality.