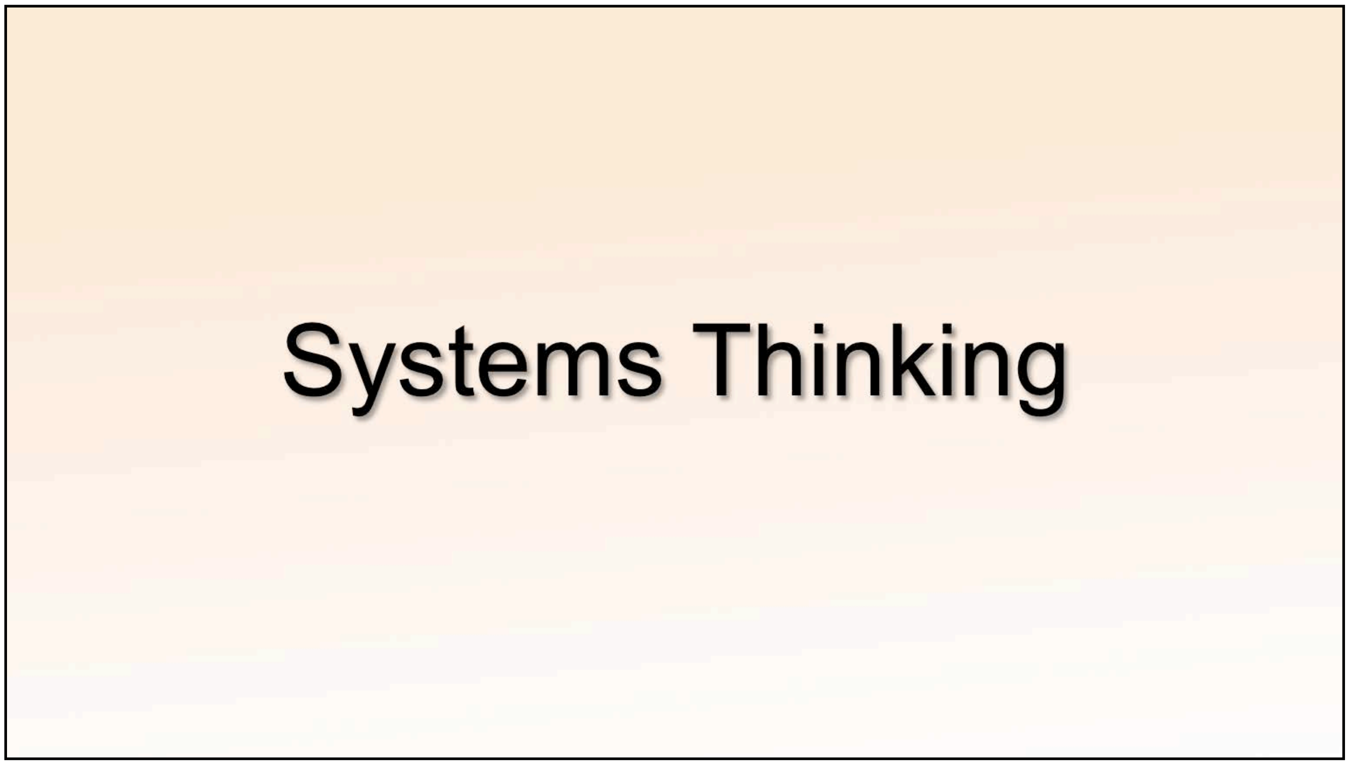
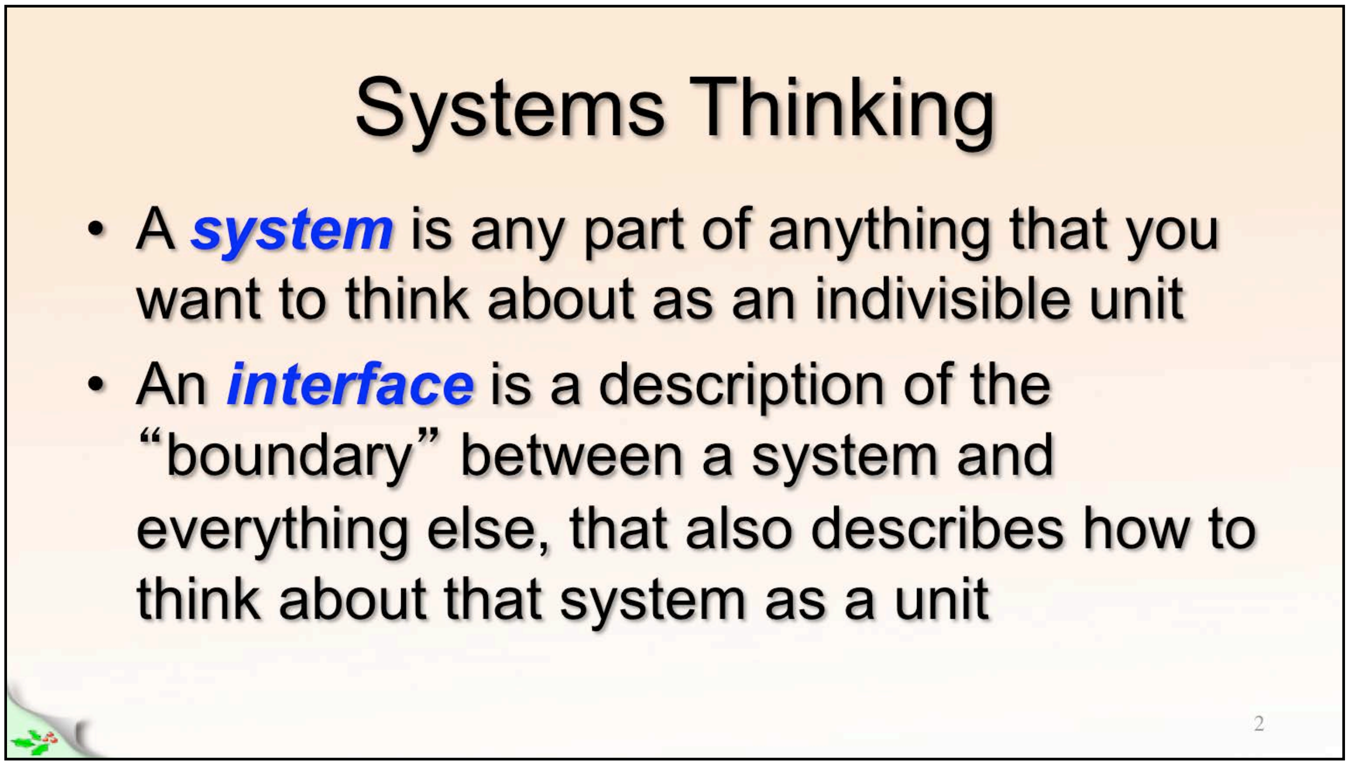


Systems Thinking

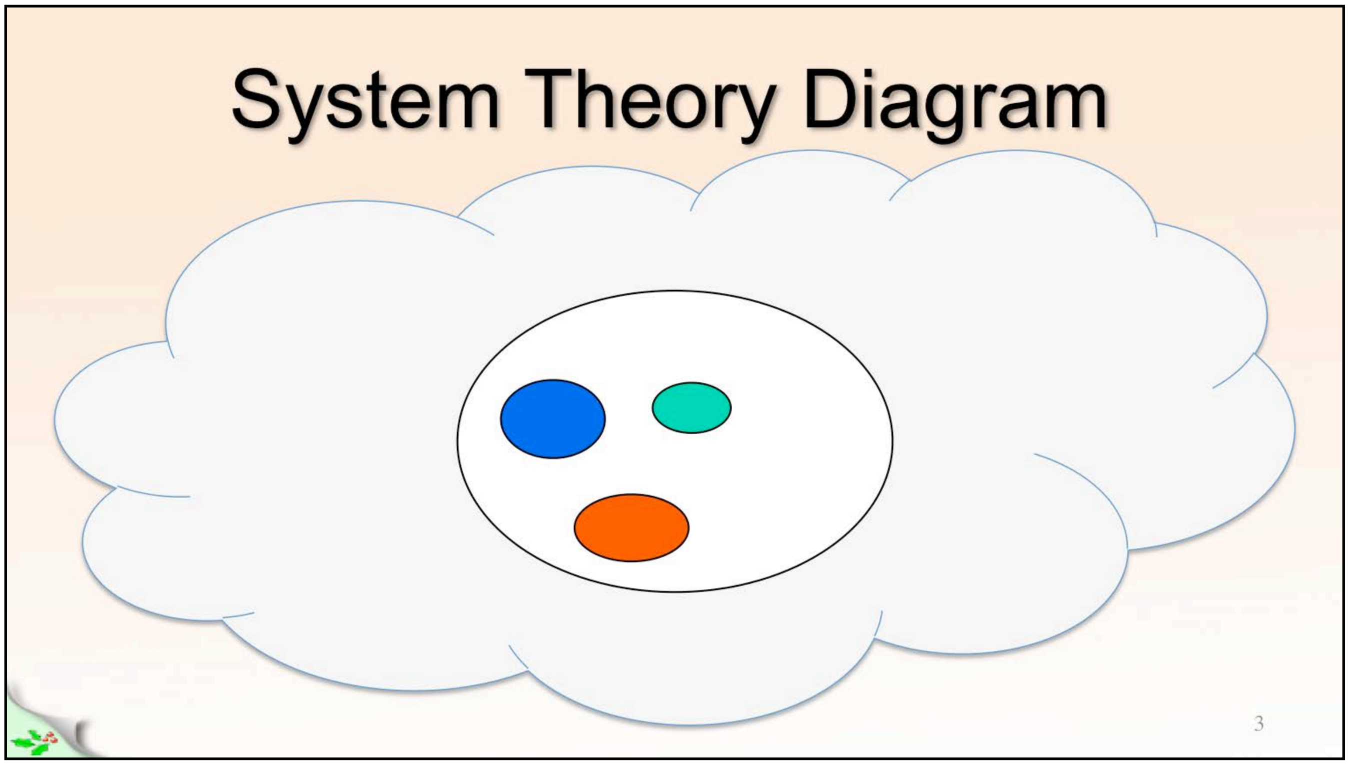
1. Hey! This is Dr. Holly.
2. Thanks for joining me for another talk from our series "Notes on Engineering Software".
3. Today's focus is on the concept called Systems Thinking.
4. And as usual we're going to be introducing a number of new concepts and terms, so be ready to pause the video and take a few notes.
5. The main idea behind Systems Thinking – is that there is an interface that separates the inside from the outside of a system. And given this interface, we want to discuss the roles played by clients and implementers of that system.
6. So let's jump right in – with Systems Thinking and get the details straightened out.



Systems Thinking

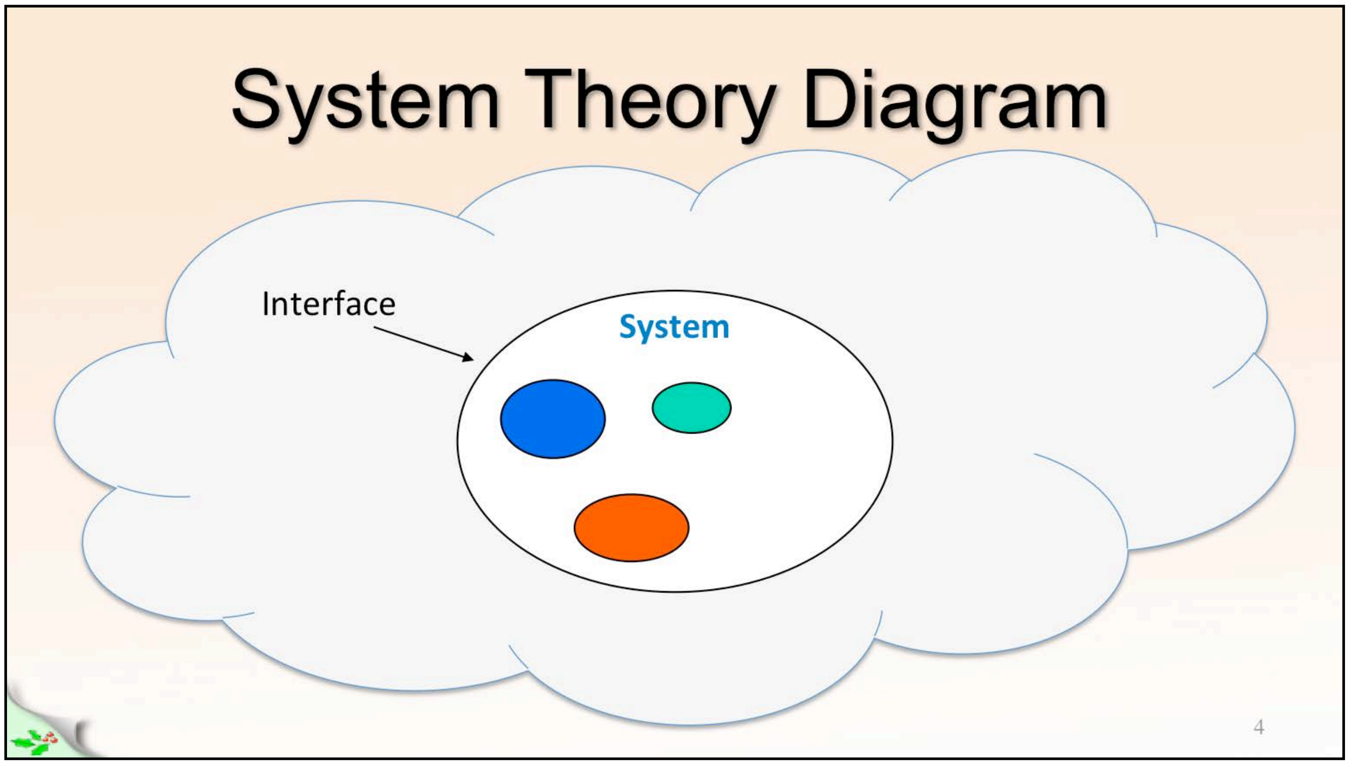
- A ***system*** is any part of anything that you want to think about as an indivisible unit
- An ***interface*** is a description of the "boundary" between a system and everything else, that also describes how to think about that system as a unit

1. In Systems Thinking – there is a System, which is anything that you want to think about as an indivisible unit.
2. Once you have established this indivisible unit, or system, you then need an interface, which is a description of the boundary between the system and everything else – and it tells you how to think or reason about the system.

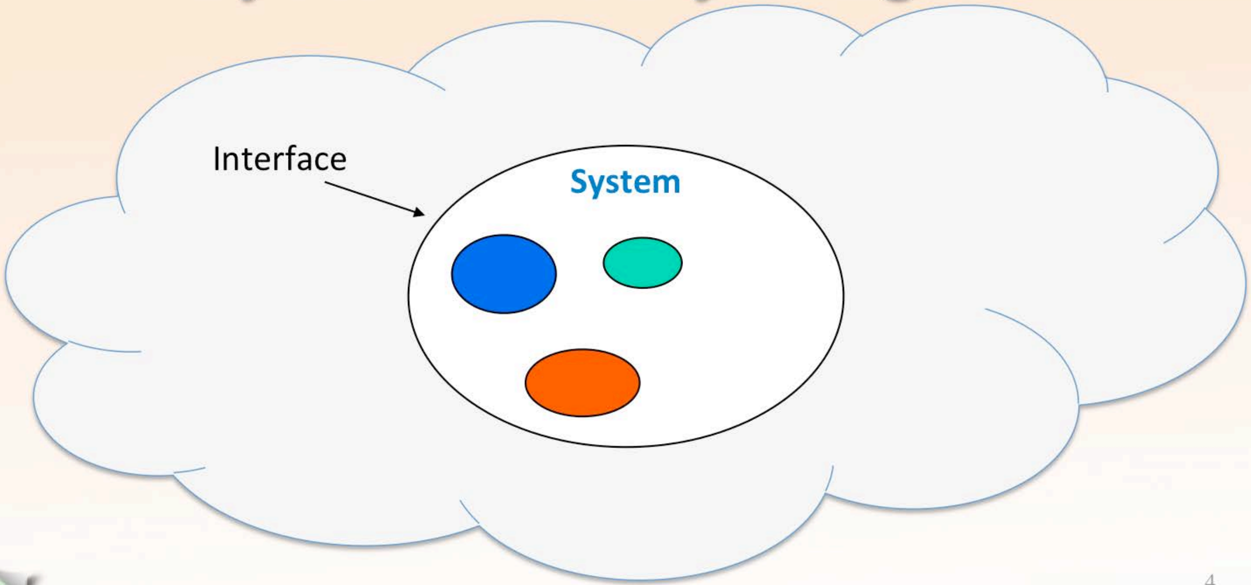


System Theory Diagram

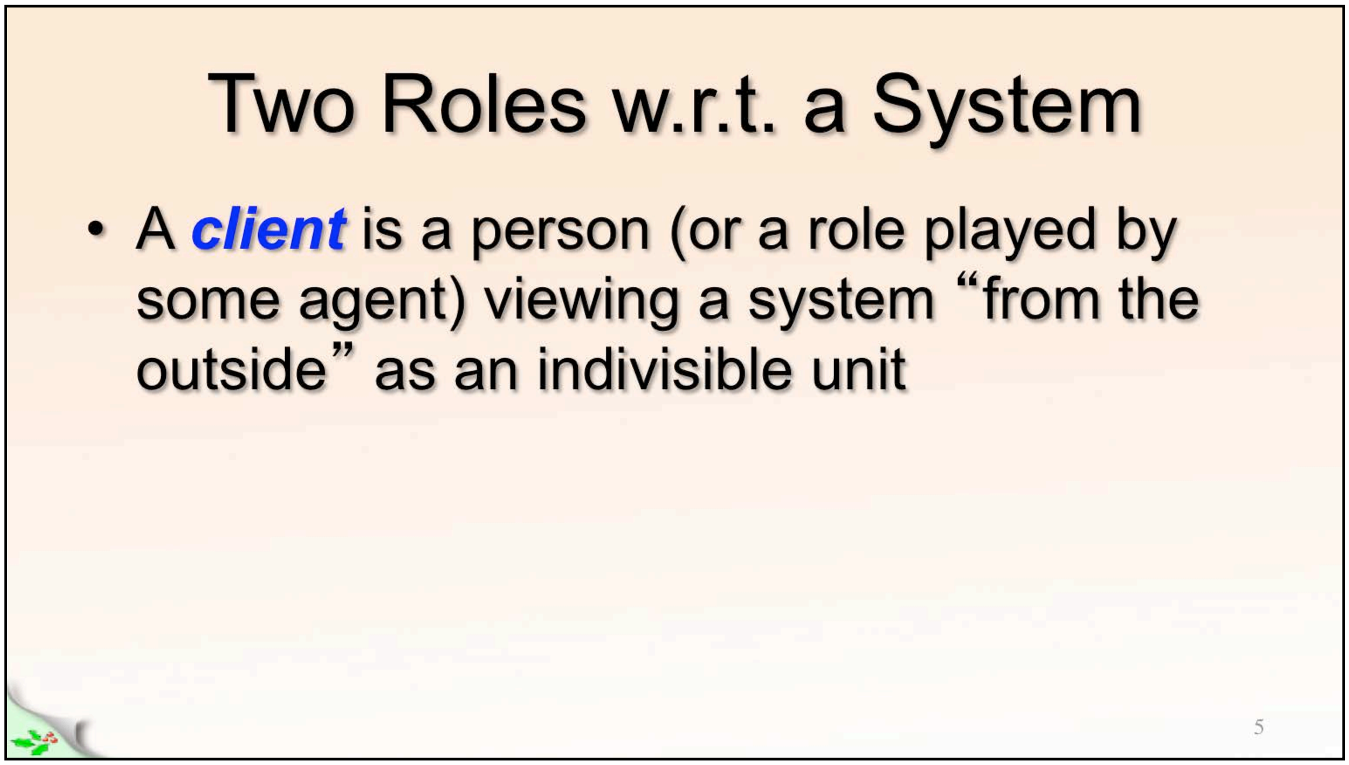
1. Let’s take a look at diagram.
2. In this diagram you see the circle in the middle.



System Theory Diagram



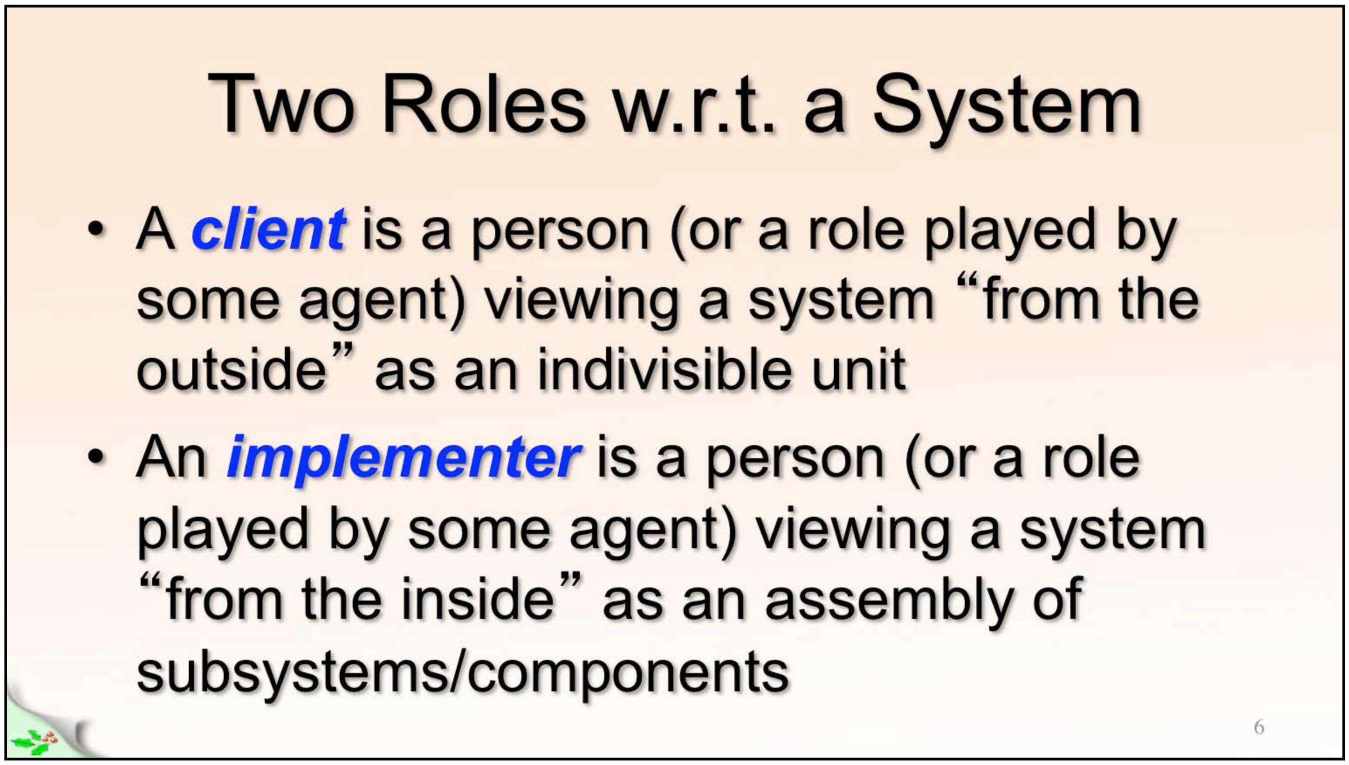
1. This represents a system, something we want to think about as an indivisible unit.
2. Once we have identified this system, we now have the interface, and outside of the interface we have everything else – which is not part of our system.
3. Building on this, we now look at two different roles to be played with respect to the system and its interface.



Two Roles w.r.t. a System

- A ***client*** is a person (or a role played by some agent) viewing a system "from the outside" as an indivisible unit

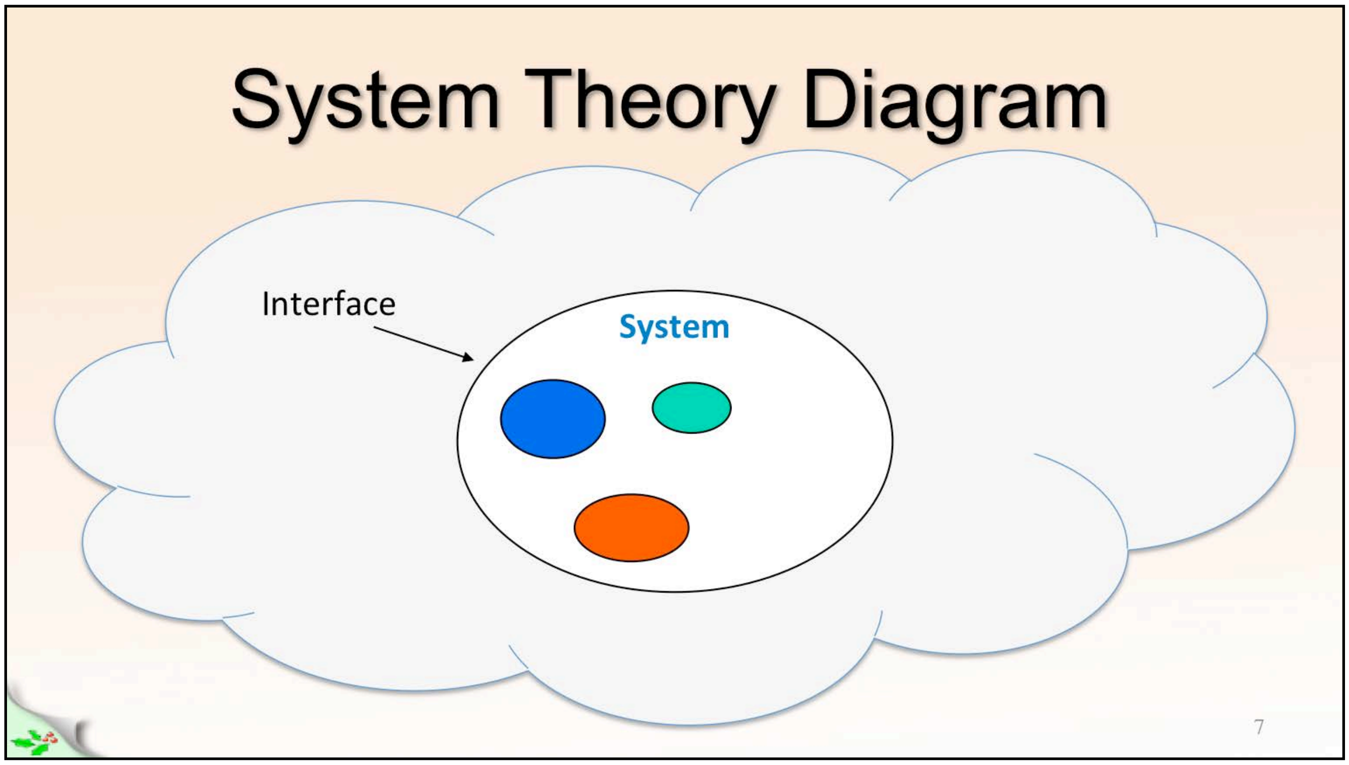
1. One role – is called the "client role", this is a person or an agent viewing a system from the outside.



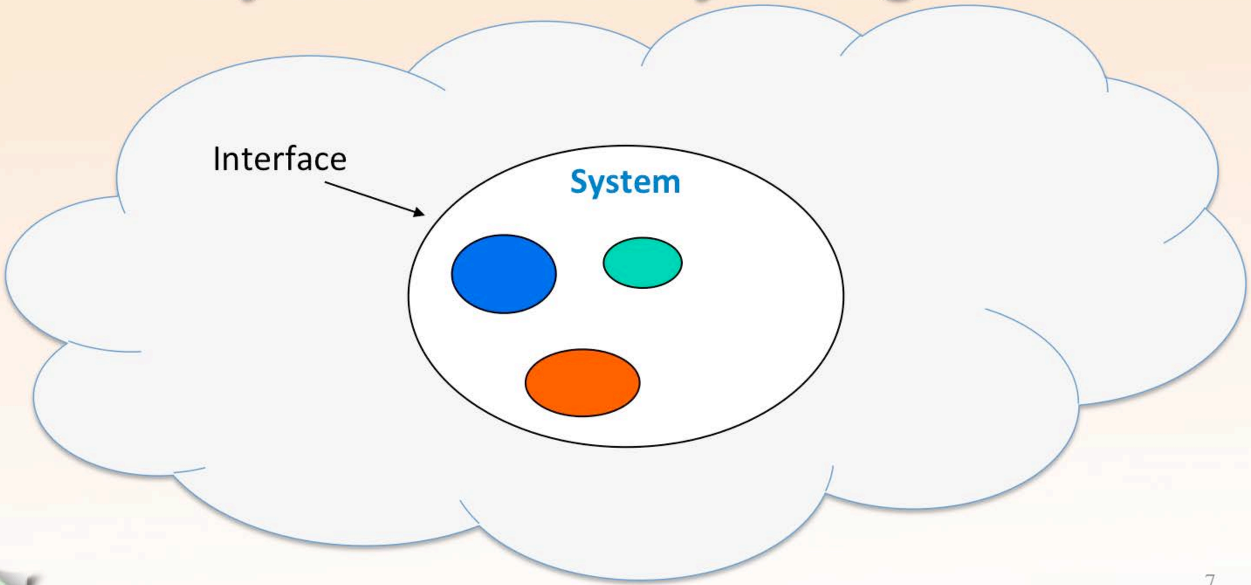
Two Roles w.r.t. a System

- A ***client*** is a person (or a role played by some agent) viewing a system “from the outside” as an indivisible unit
- An ***implementer*** is a person (or a role played by some agent) viewing a system “from the inside” as an assembly of subsystems/components

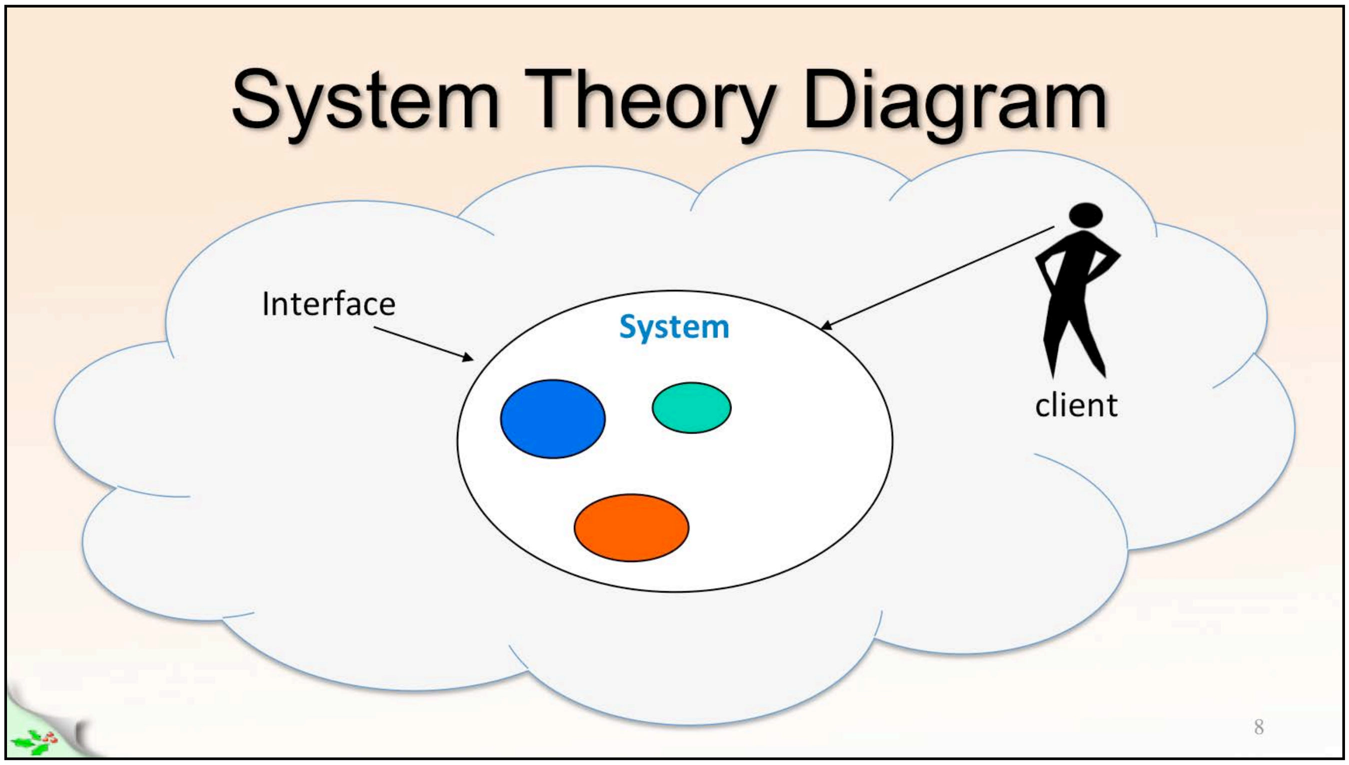
1. The second role – is called the “implementer role”, and this is a person or agent that views the system from the inside as an assembly of subsystems, or components.



System Theory Diagram

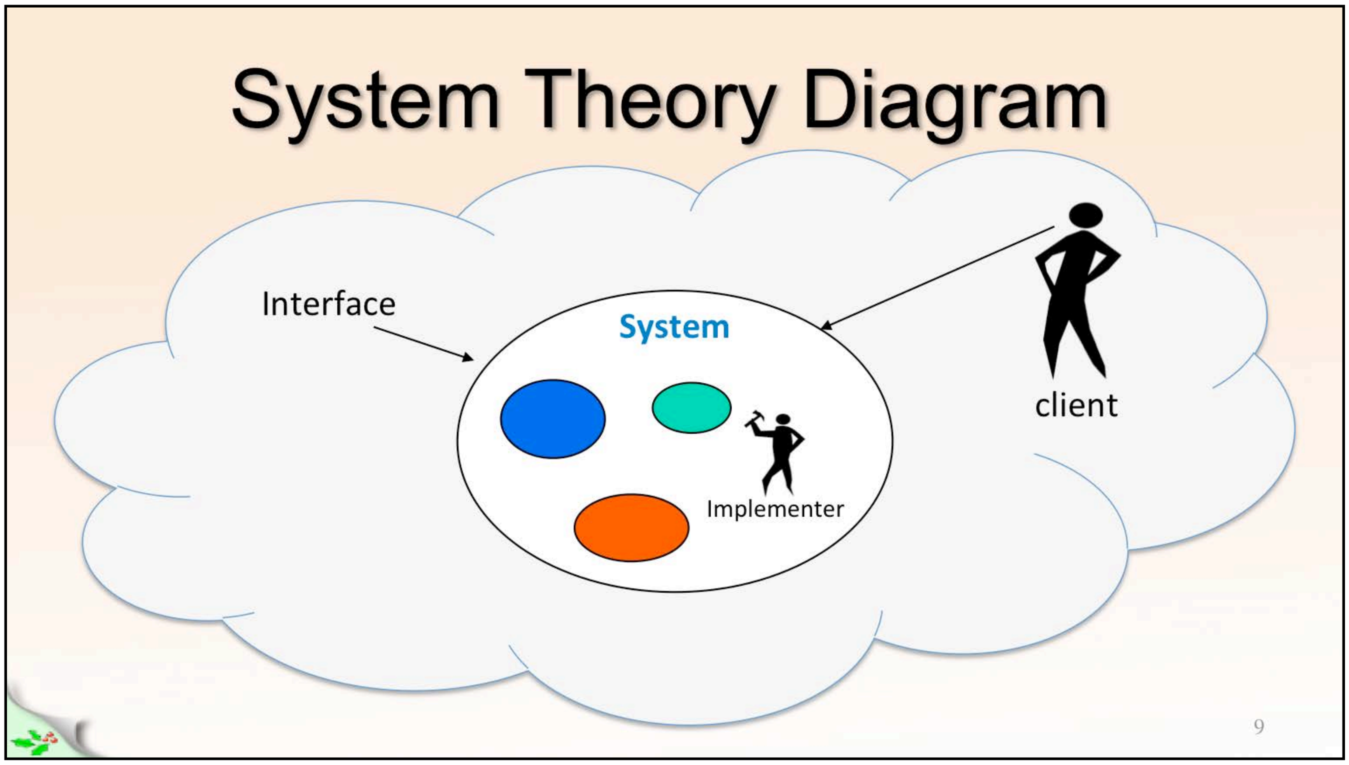


1. Let’s go back to the diagram – and modify it – so that it shows these two roles



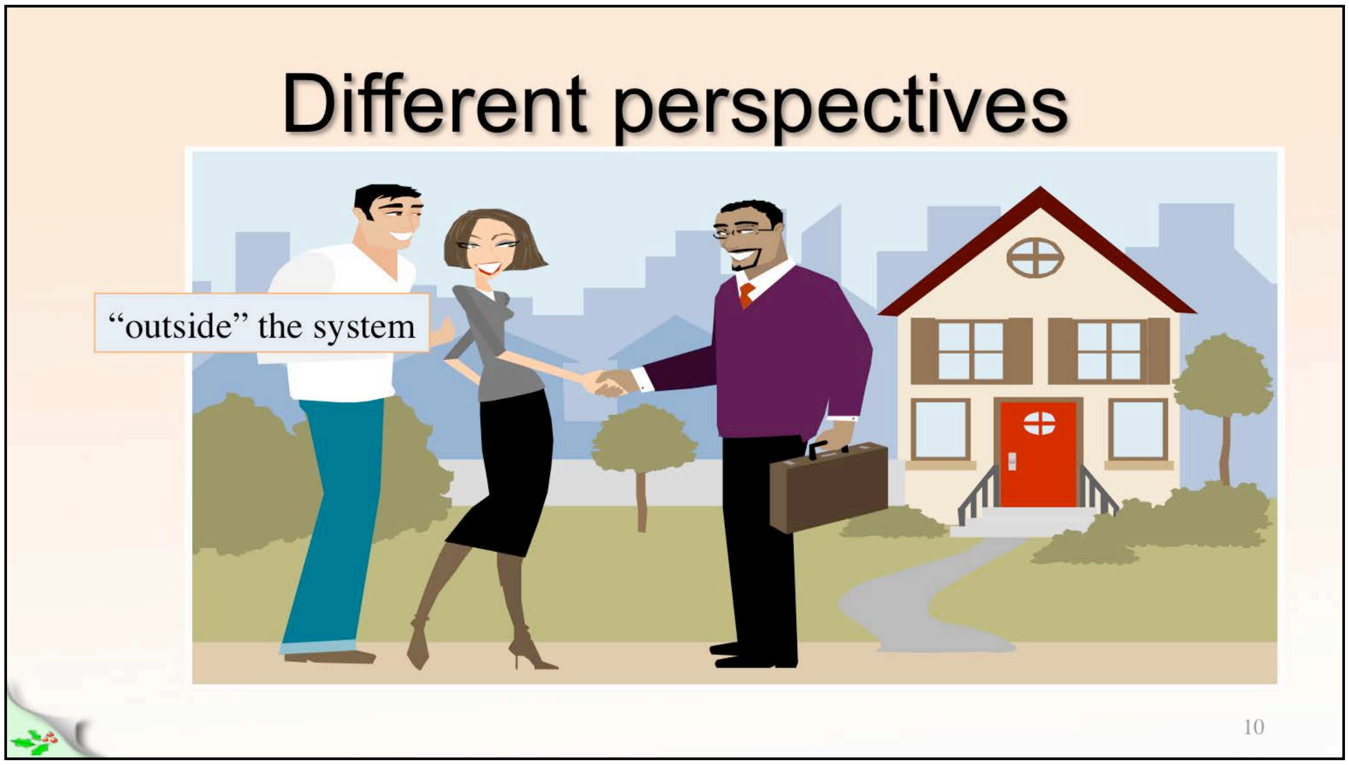
System Theory Diagram

1. Here we see the client – viewing the system and its interface from the outside



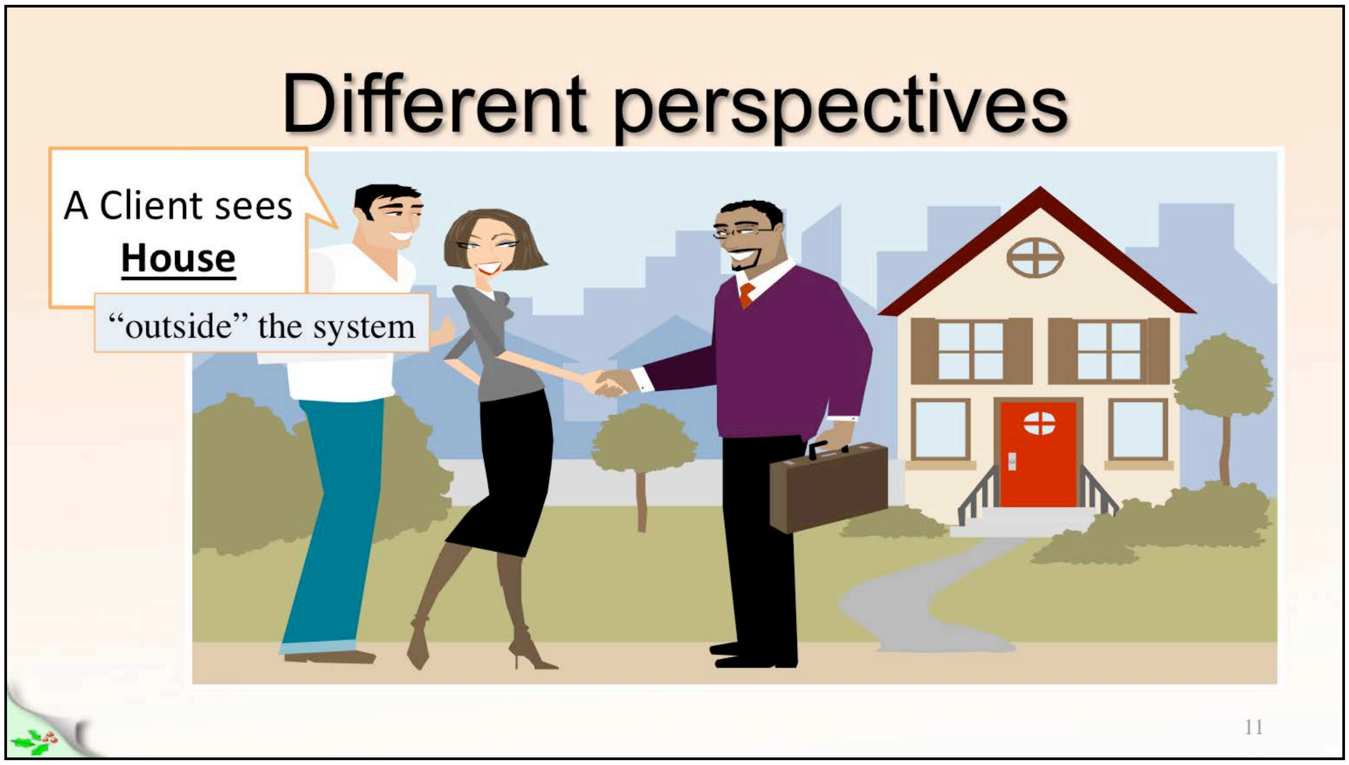
System Theory Diagram

1. And the implementer viewing the system from the inside.



Different perspectives

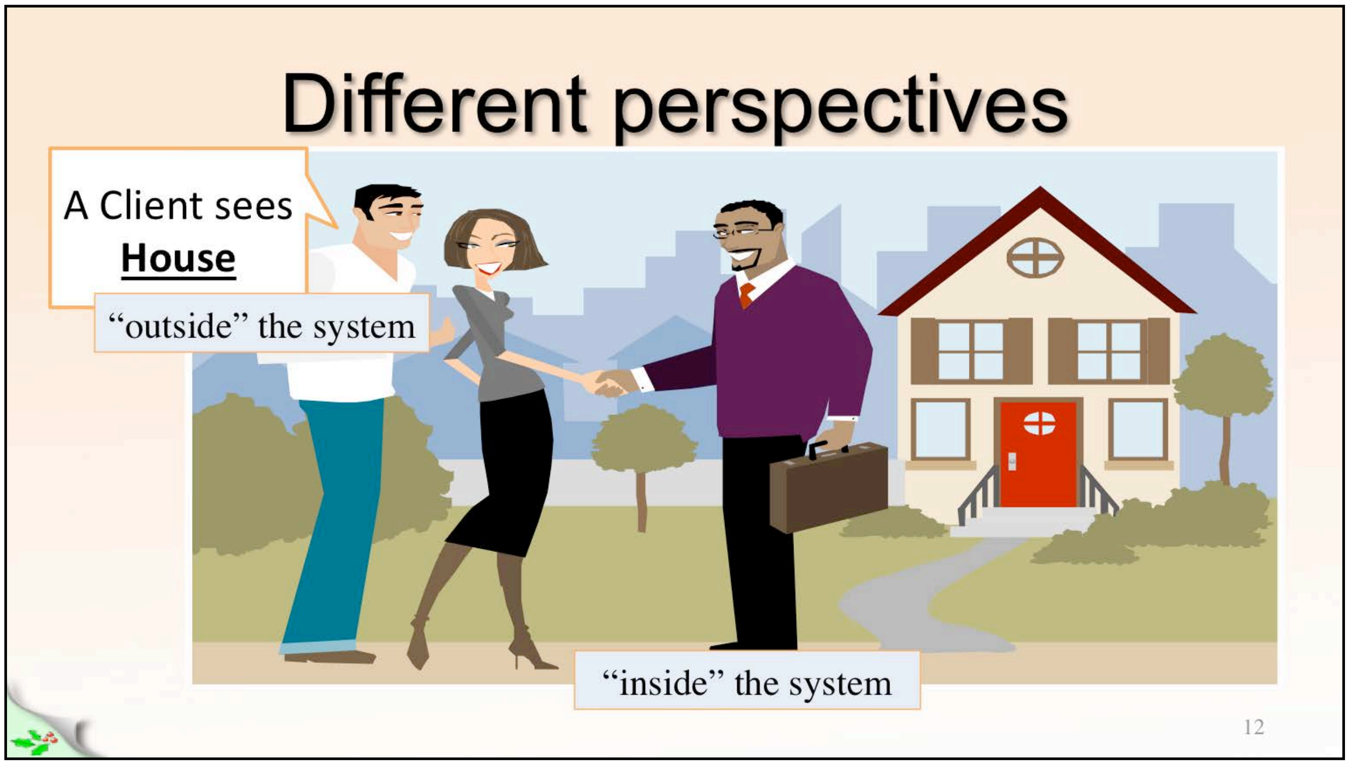
1. Now, let’s look at the perspectives of people involved in an analogous situation of buying a house.
2. On the outside of the system …



Different perspectives

A Client sees **House**
“outside” the system

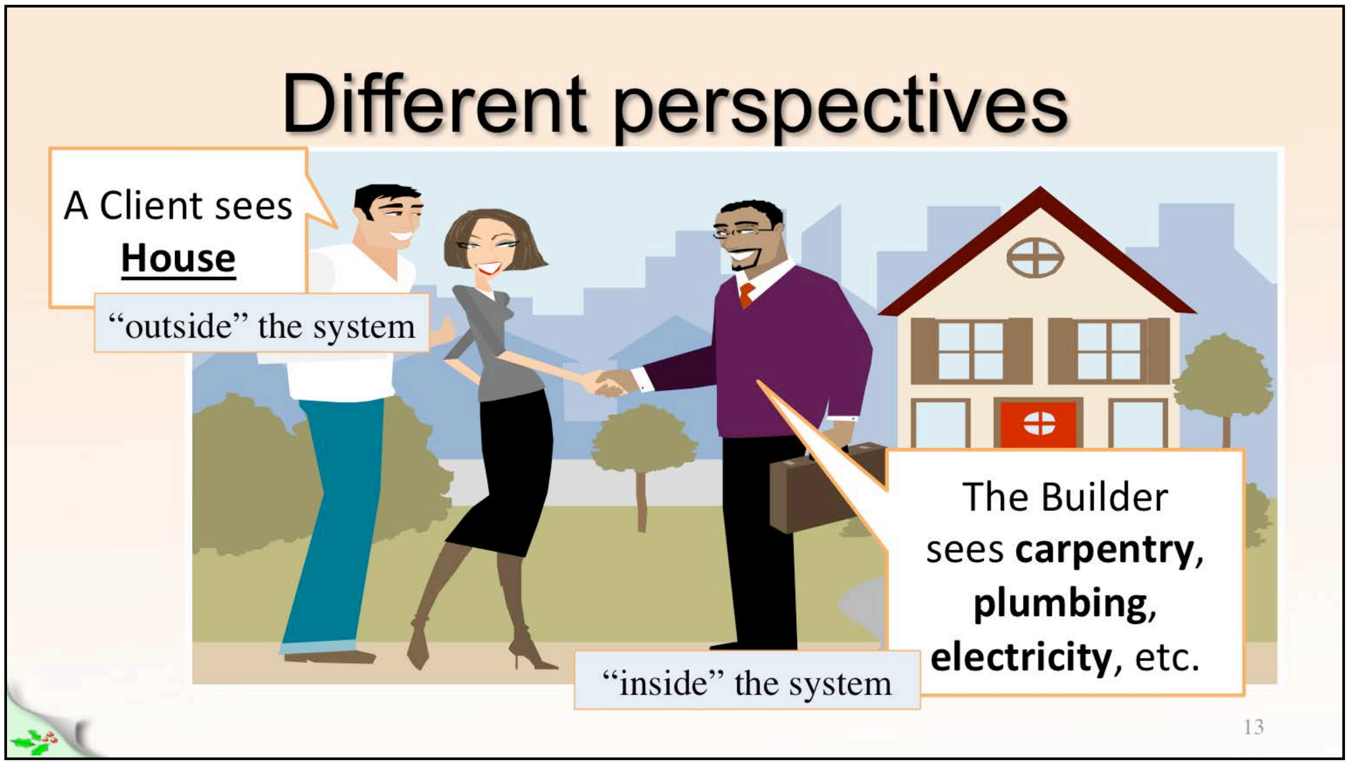
1. We have a client who considers buying the house



Different perspectives

A Client sees **House**
"outside" the system
"inside" the system

1. On the inside of the system …



Different perspectives

A Client sees **House**

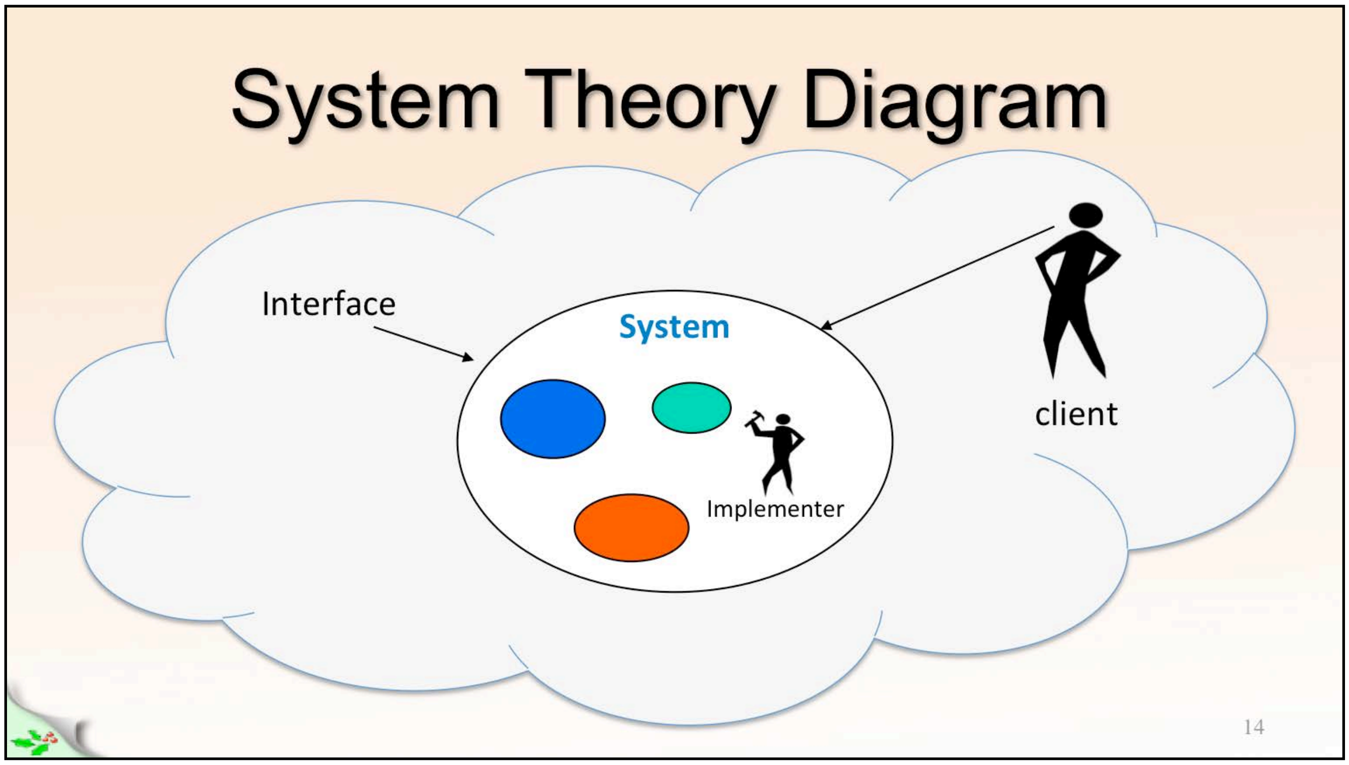
"outside" the system

The Builder sees **carpentry**, **plumbing**, **electricity**, etc.

"inside" the system

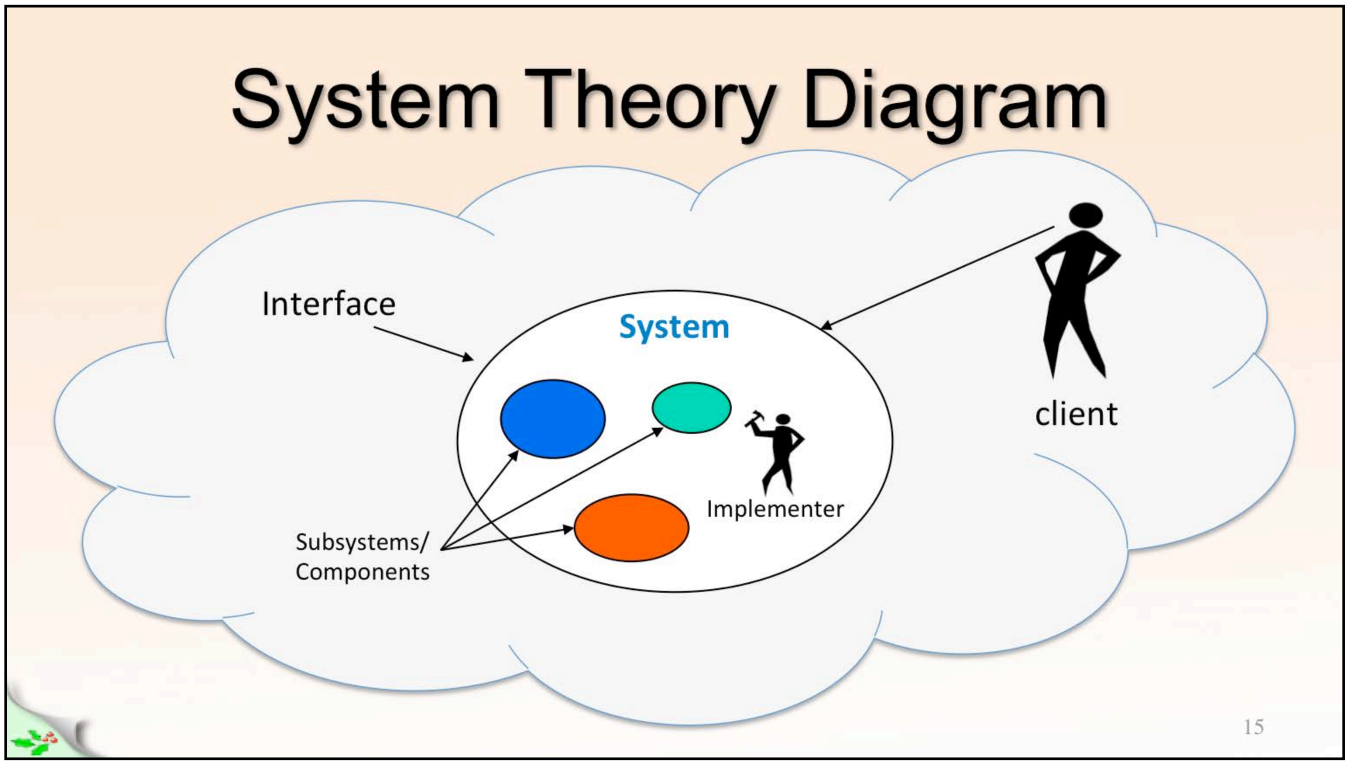
13

1. We have the builder who is concerned with the construction details of the house.



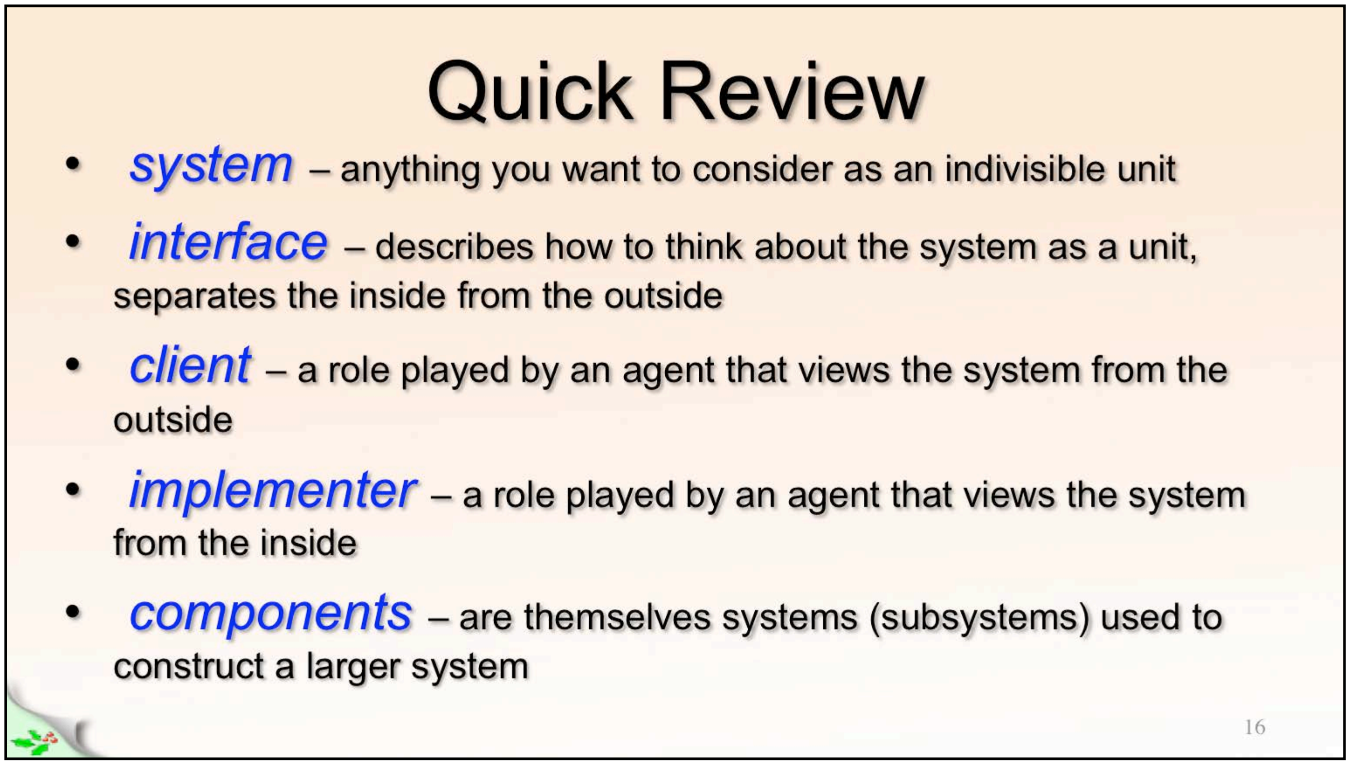
System Theory Diagram

1. STUDENT – But professor, I have a question, on that System Theory Diagram there are three circles on the inside of the system, what are those?



System Theory Diagram

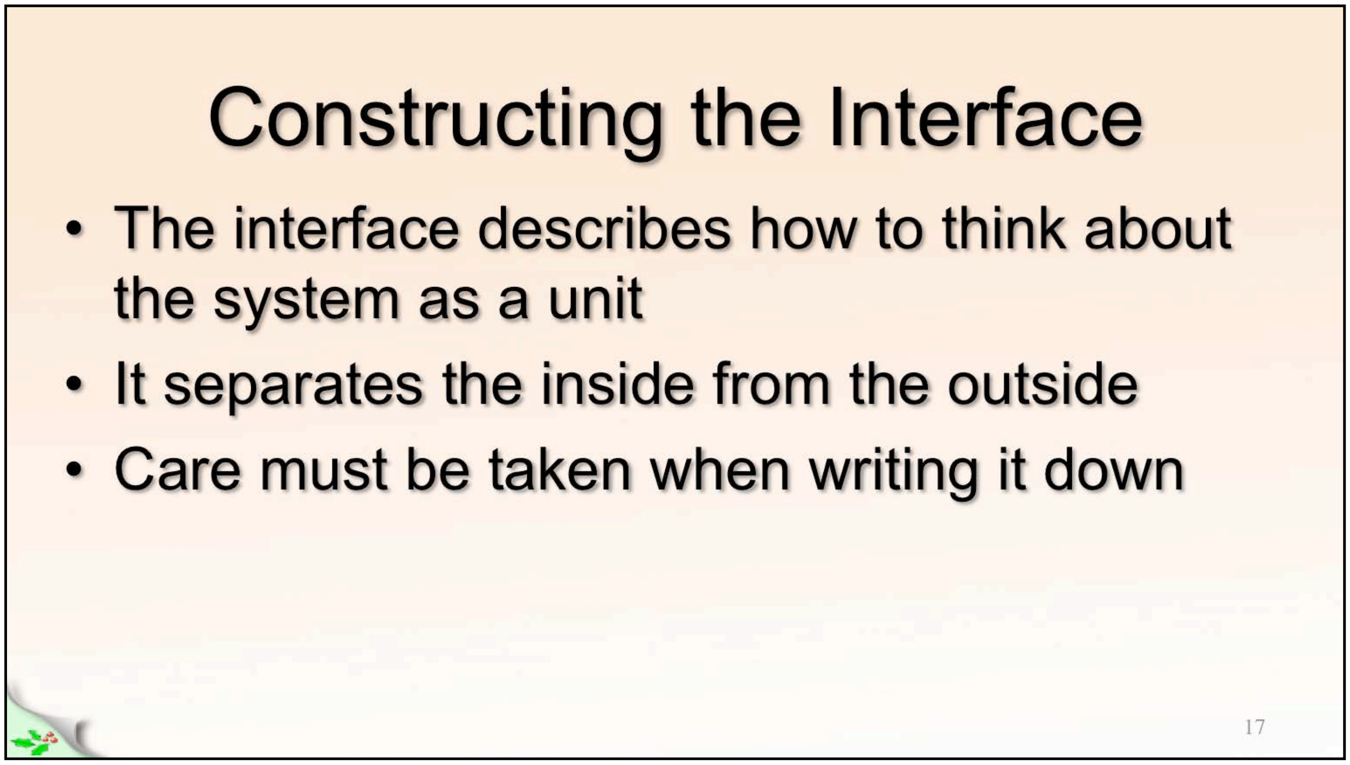
1. Yeah, good question. Earlier I mentioned – that when building the system – the implementer works with components that are internal to the system.
2. These components – are really just other previously built systems. Sometimes they are called subsystems.



Quick Review

- *system* – anything you want to consider as an indivisible unit
- *interface* – describes how to think about the system as a unit, separates the inside from the outside
- *client* – a role played by an agent that views the system from the outside
- *implementer* – a role played by an agent that views the system from the inside
- *components* – are themselves systems (subsystems) used to construct a larger system

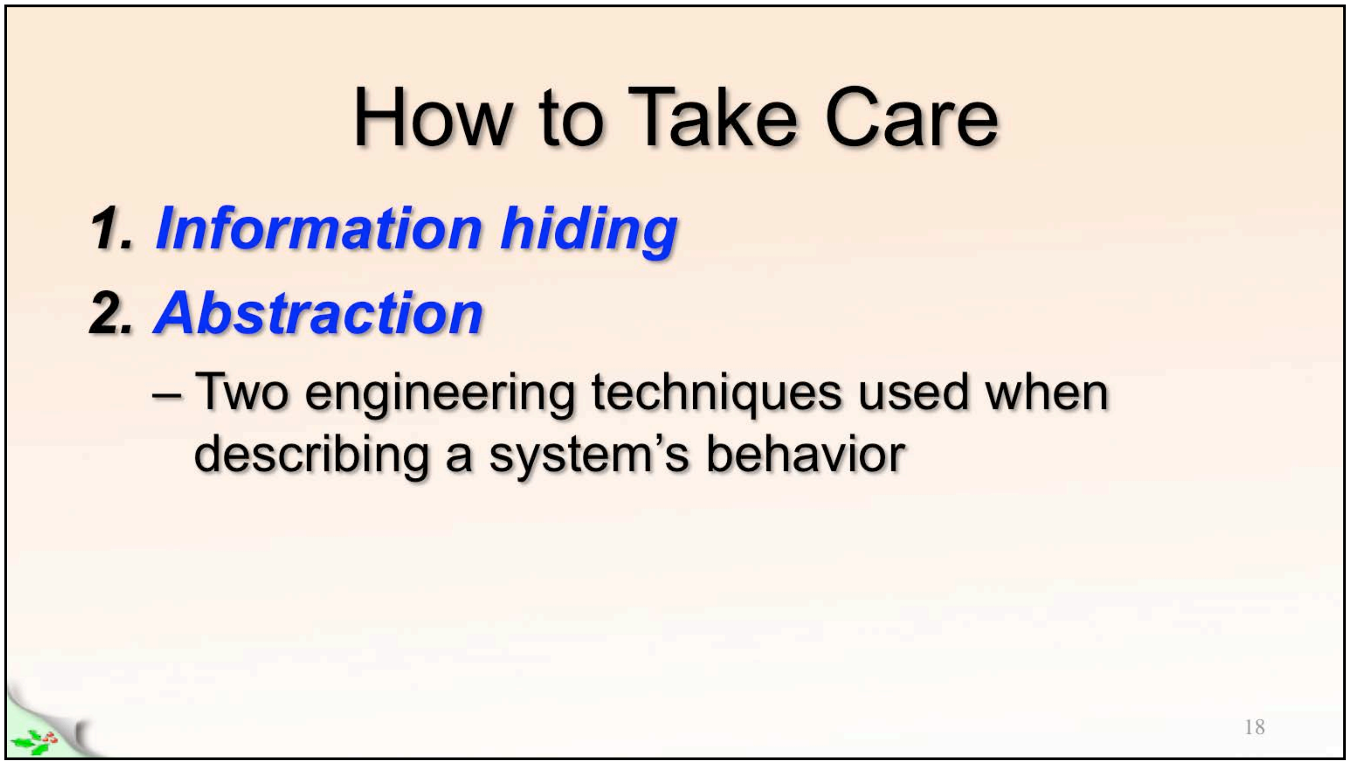
1. Now, let's do a quick review of the concepts and terms we've looked at so far.
2. It all starts with considering some item as an indivisible unit, this is our system.
3. Once we've established a system, this gives rise to an interface that separates the inside from the outside of the system.
4. And here's something that's really important, the interface describes how to think about the system as an indivisible unit.
5. Given this interface, we now have two roles, the client and the implementer.
6. The client – views the system from the outside – and uses the details found in the interface to think about how to use the system.
7. The implementer – views the system from the inside, and uses other components to implement the system.
8. Additionally, the implementer utilizes the description in the interface in order to understand what needs to be done in order to correctly implement the system.
9. These internal components – are themselves, systems – each with their own interface. And the implementer,
10. acts as a client of these internal components.



Constructing the Interface

- The interface describes how to think about the system as a unit
- It separates the inside from the outside
- Care must be taken when writing it down

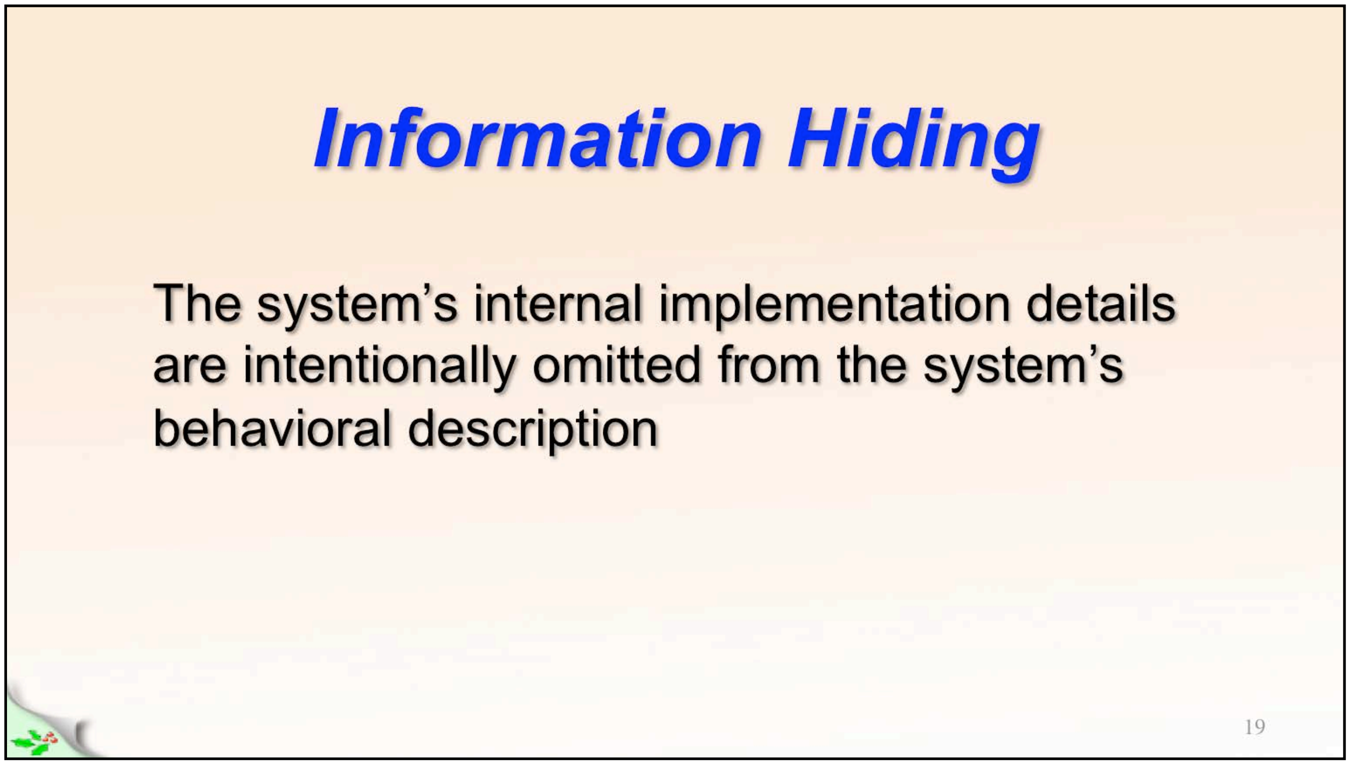
1. At this point we need to shift gears, and talk about how to construct the interface.
2. As we have said before, the interface describes how to think about the system as a unit as well as separating the inside from the outside.
3. We must be very careful – when writing down the details of the interface, because both the client and the implementer depend on the interface for their understanding of how the system should work.



How to Take Care

1. ***Information hiding***
2. ***Abstraction***
   - Two engineering techniques used when describing a system's behavior

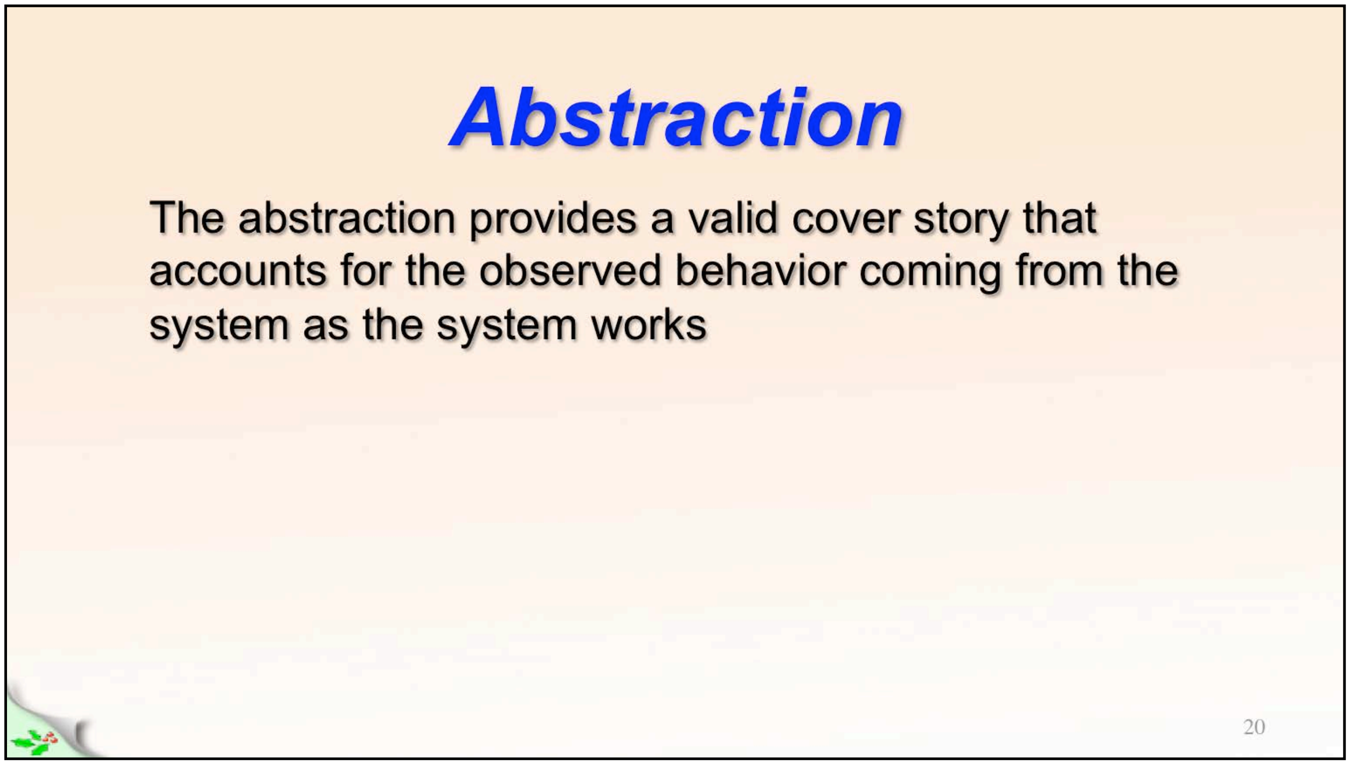
1. Here are two fundamental engineering techniques that help you “be careful” when writing down the interface of a system.
2. Information hiding
3. And abstraction.
4. Let’s take a closer look at these two techniques.



Information Hiding

The system's internal implementation details are intentionally omitted from the system's behavioral description

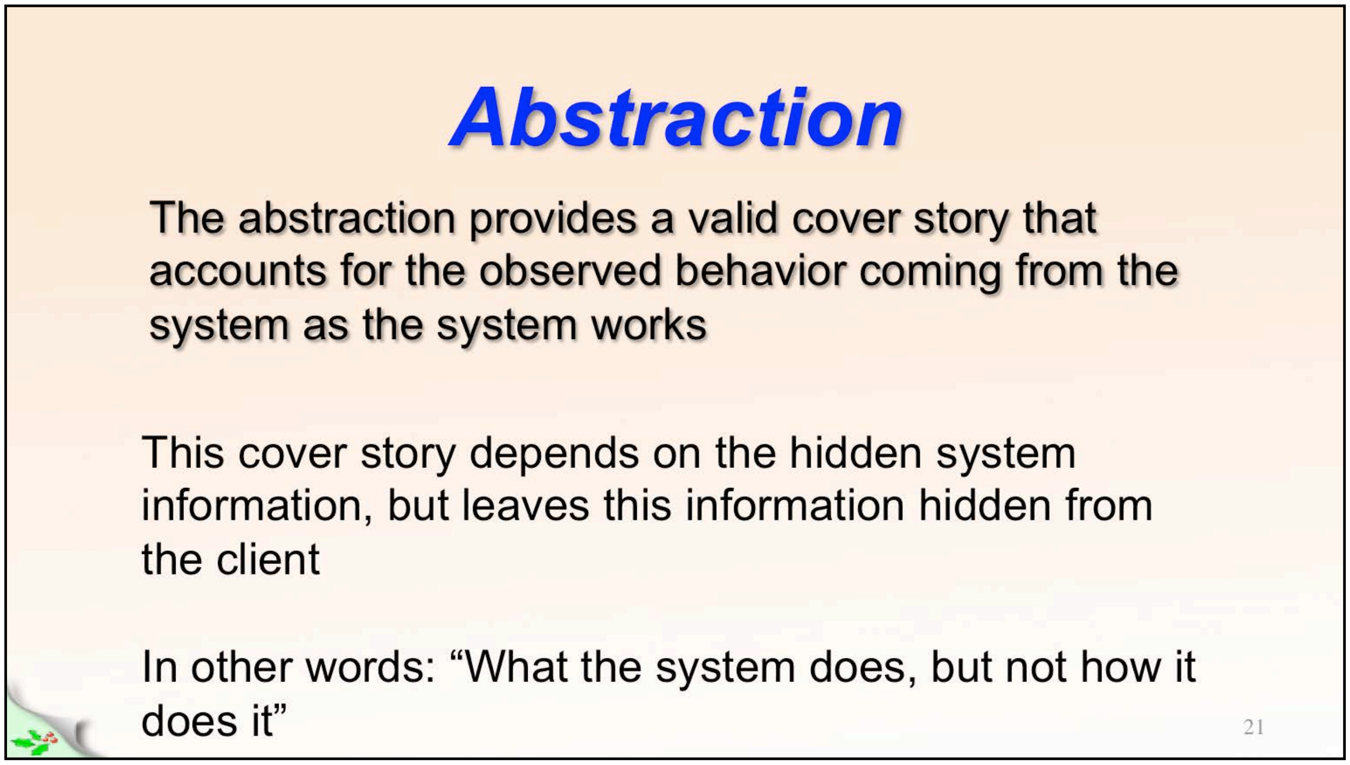
1. With information hiding, we deliberately omit internal implementation details from the system's behavioral description.
2. These internal details are not needed by the client in order to understand what the system does or what services the system can perform for the client.



Abstraction

The abstraction provides a valid cover story that accounts for the observed behavior coming from the system as the system works

1. What abstraction does, is provide a “valid cover story” that accounts for observable behavior emanating from the system.
2. Part of what makes it a, “valid” cover story is that it allows the client, to predict the behavior of the system given a particular set of circumstances.



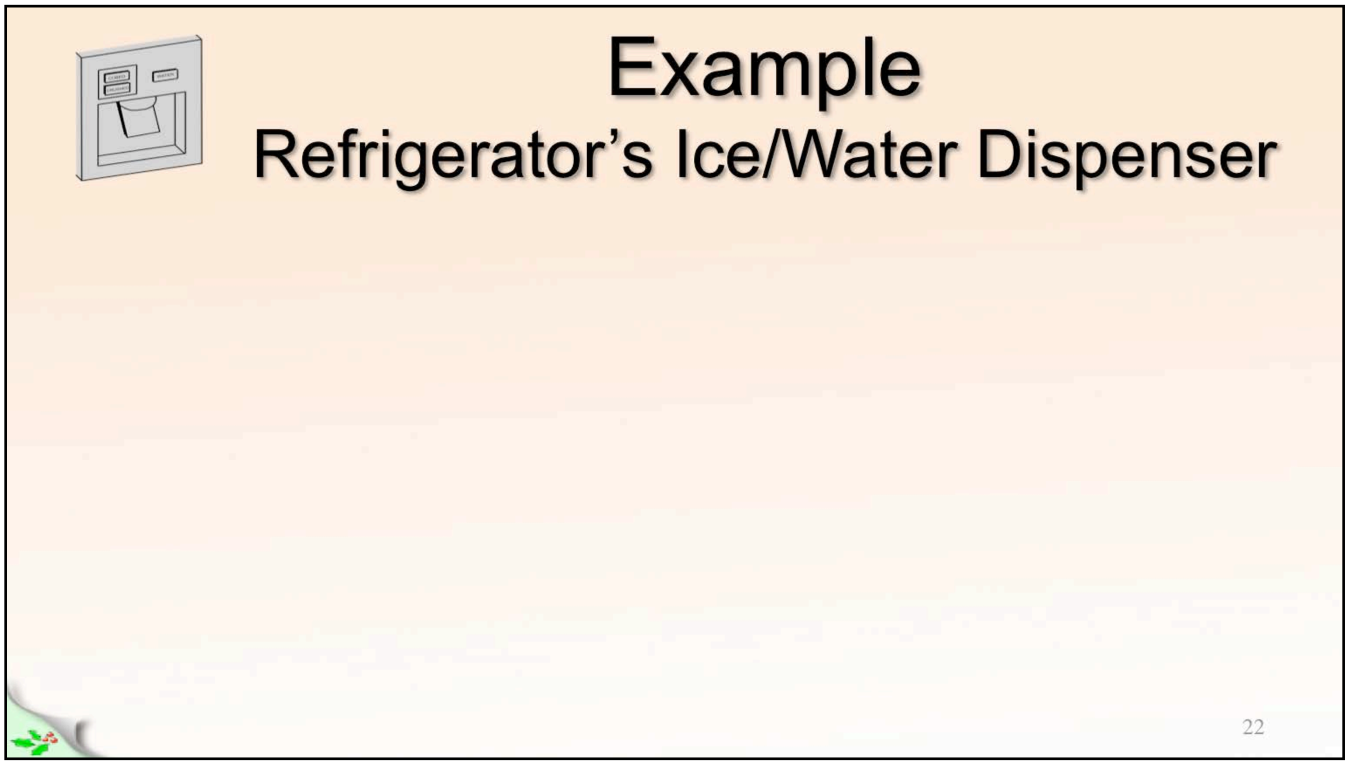
Abstraction

The abstraction provides a valid cover story that accounts for the observed behavior coming from the system as the system works

This cover story depends on the hidden system information, but leaves this information hidden from the client

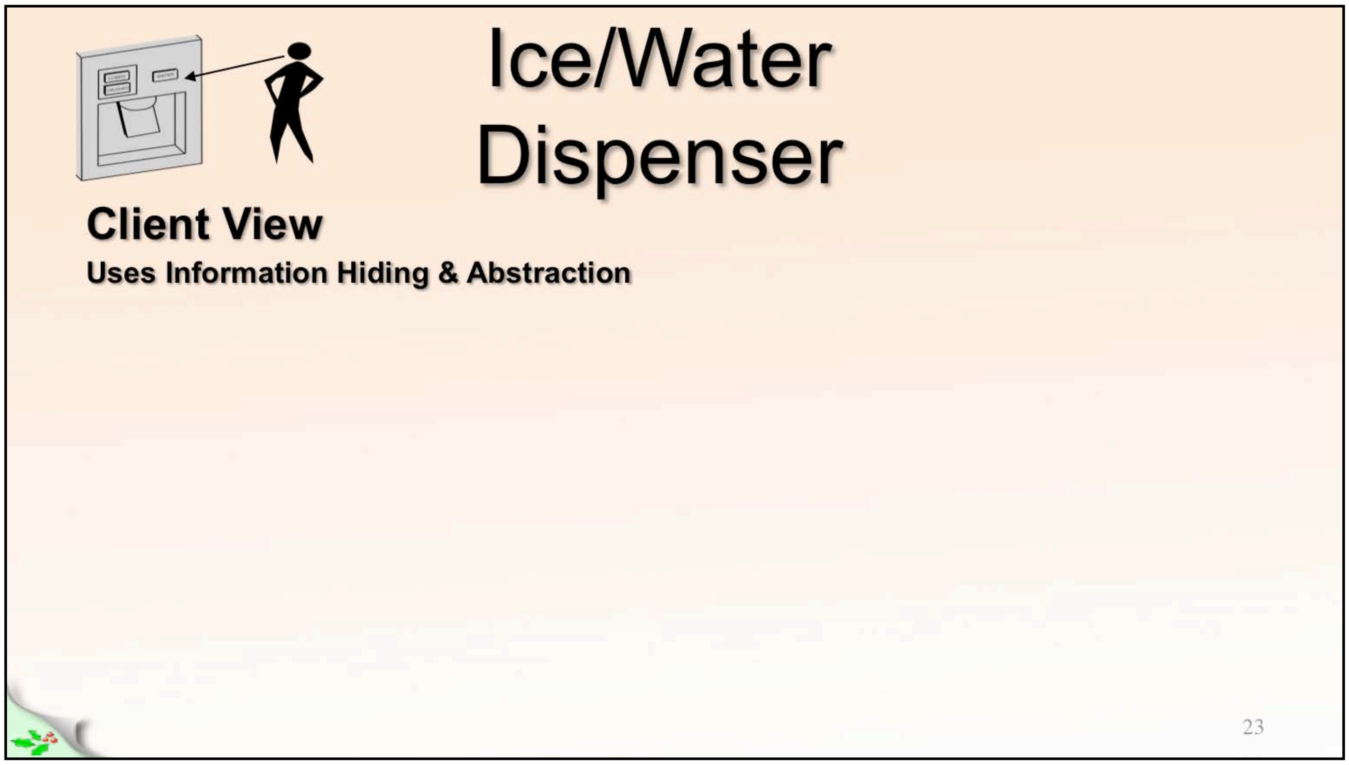
In other words: “What the system does, but not how it does it”

1. In order for the cover story to provide a valid explanation in all situations, it must take into account how the internal components function, but at the same time it must leave these components hidden from the client
2. In other words the cover story must describe: “What the system does, but not how it does it”



Example
Refrigerator's Ice/Water Dispenser

1. Let's look at an example of an everyday appliance.
2. A refrigerator's ice and water dispenser

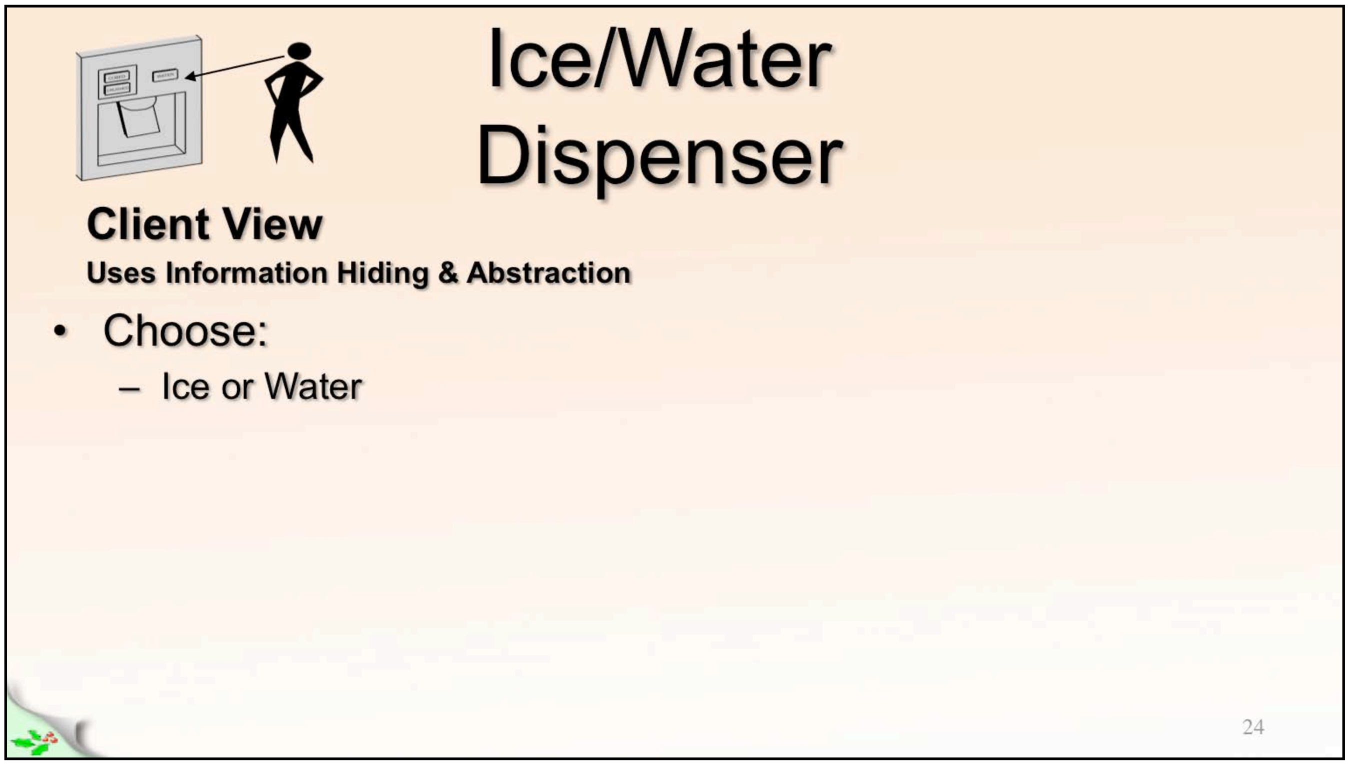


Ice/Water Dispenser

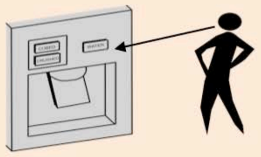
Client View

Uses Information Hiding & Abstraction

1. To create the client's view, information hiding and abstraction are used.



Ice/Water Dispenser

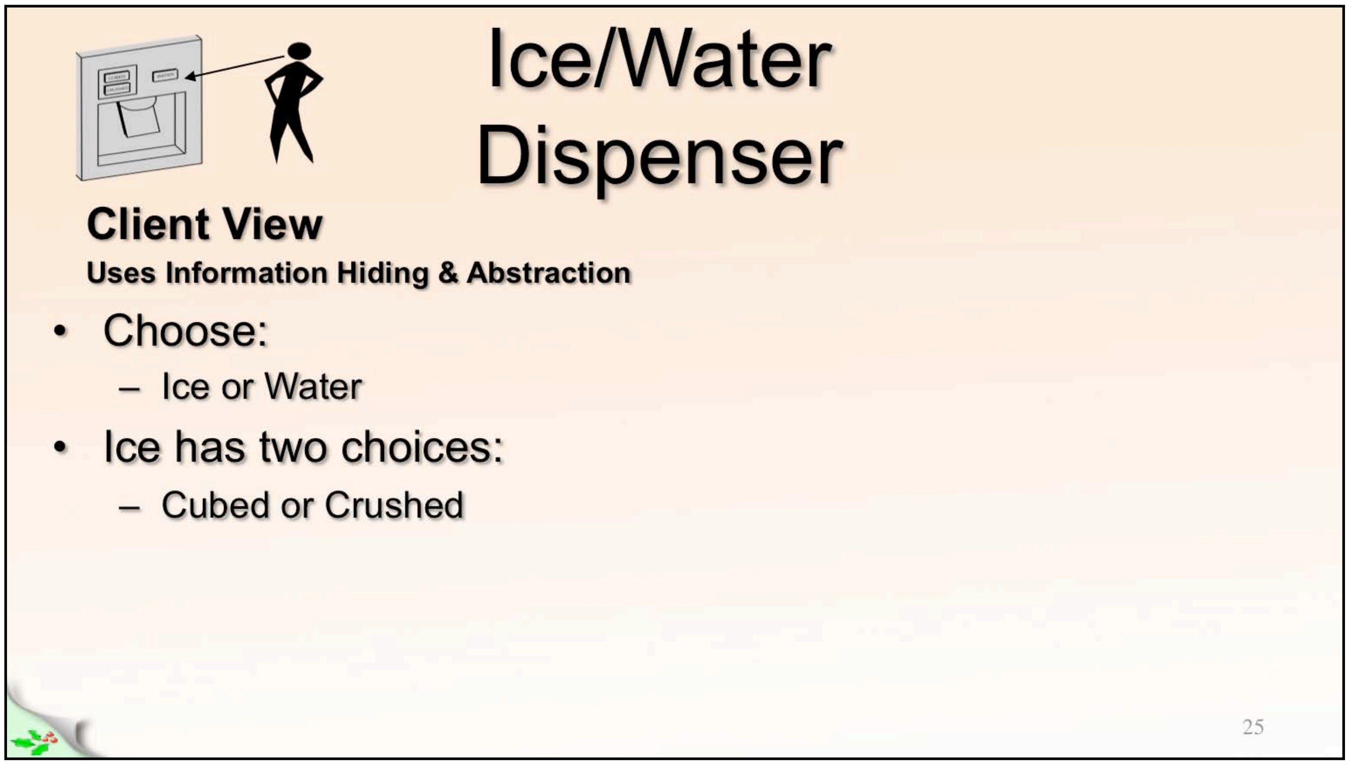


Client View

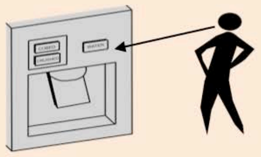
Uses Information Hiding & Abstraction

- Choose:
  - Ice or Water

1. The client, can choose between Ice and Water



Ice/Water Dispenser



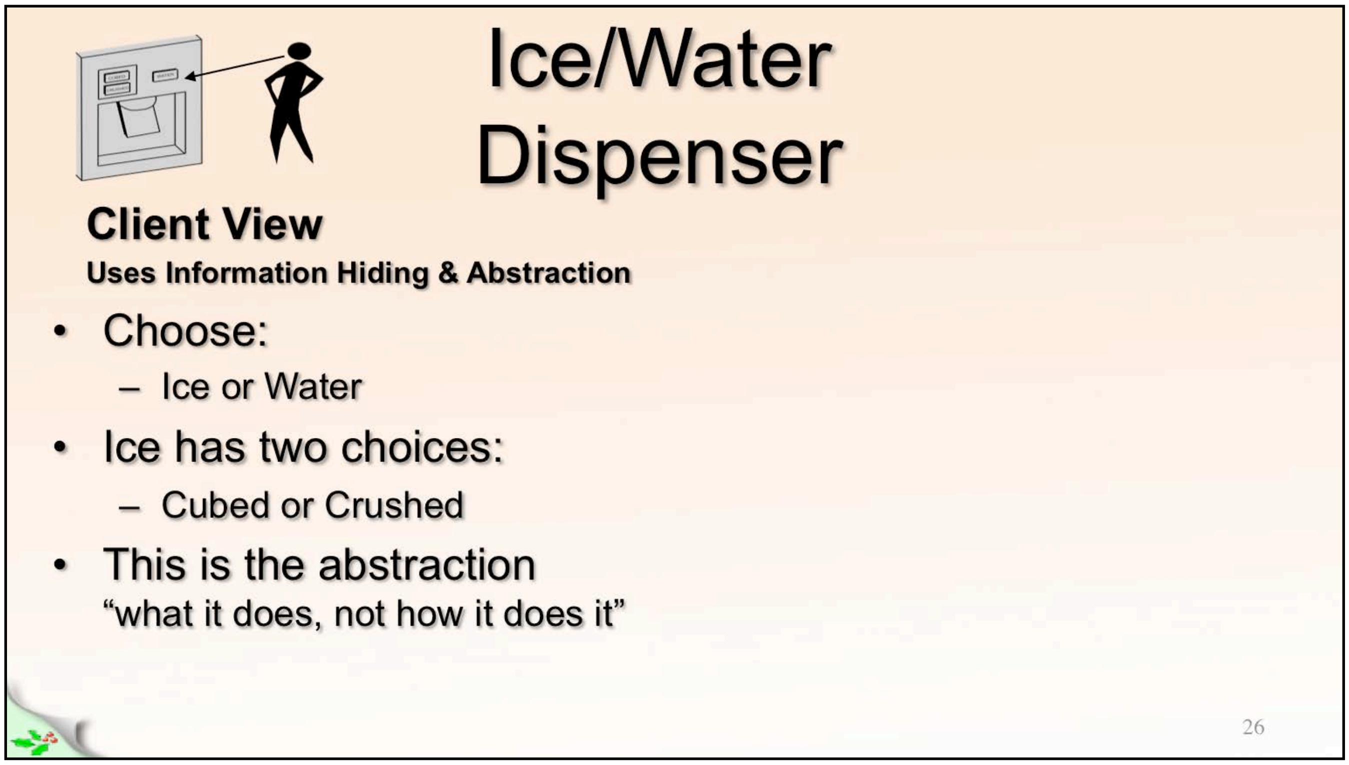
Client View

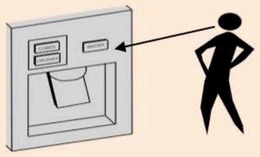
Uses Information Hiding & Abstraction

- Choose:
  - Ice or Water
- Ice has two choices:
  - Cubed or Crushed

25

1. If the client chooses Ice, then there are two alternatives, cubed or crushed





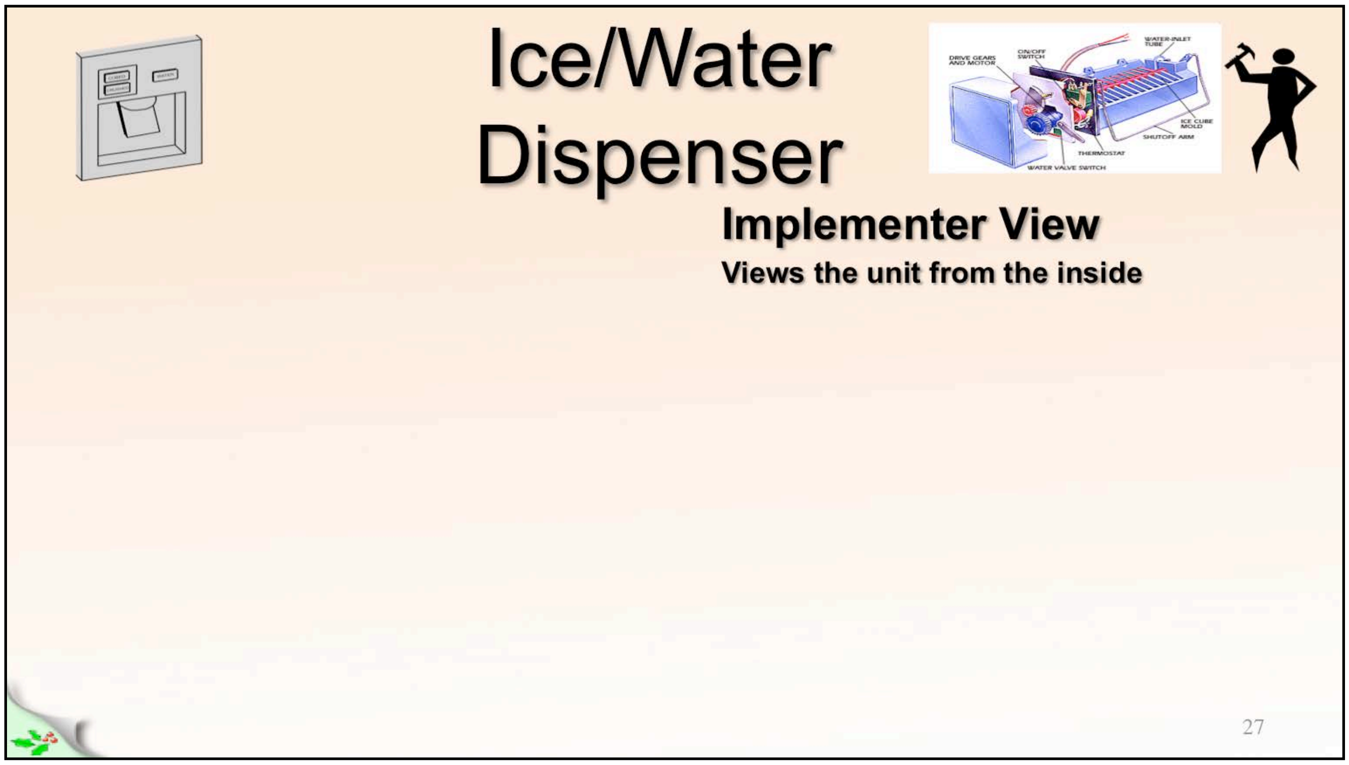
Ice/Water Dispenser

Client View

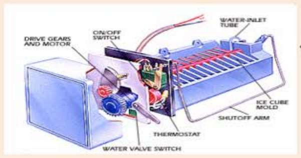
Uses Information Hiding & Abstraction

- Choose:
  - Ice or Water
- Ice has two choices:
  - Cubed or Crushed
- This is the abstraction
  "what it does, not how it does it"

1. Here, abstraction provides a description of what the Ice & Water Dispenser does, but not how it does it.



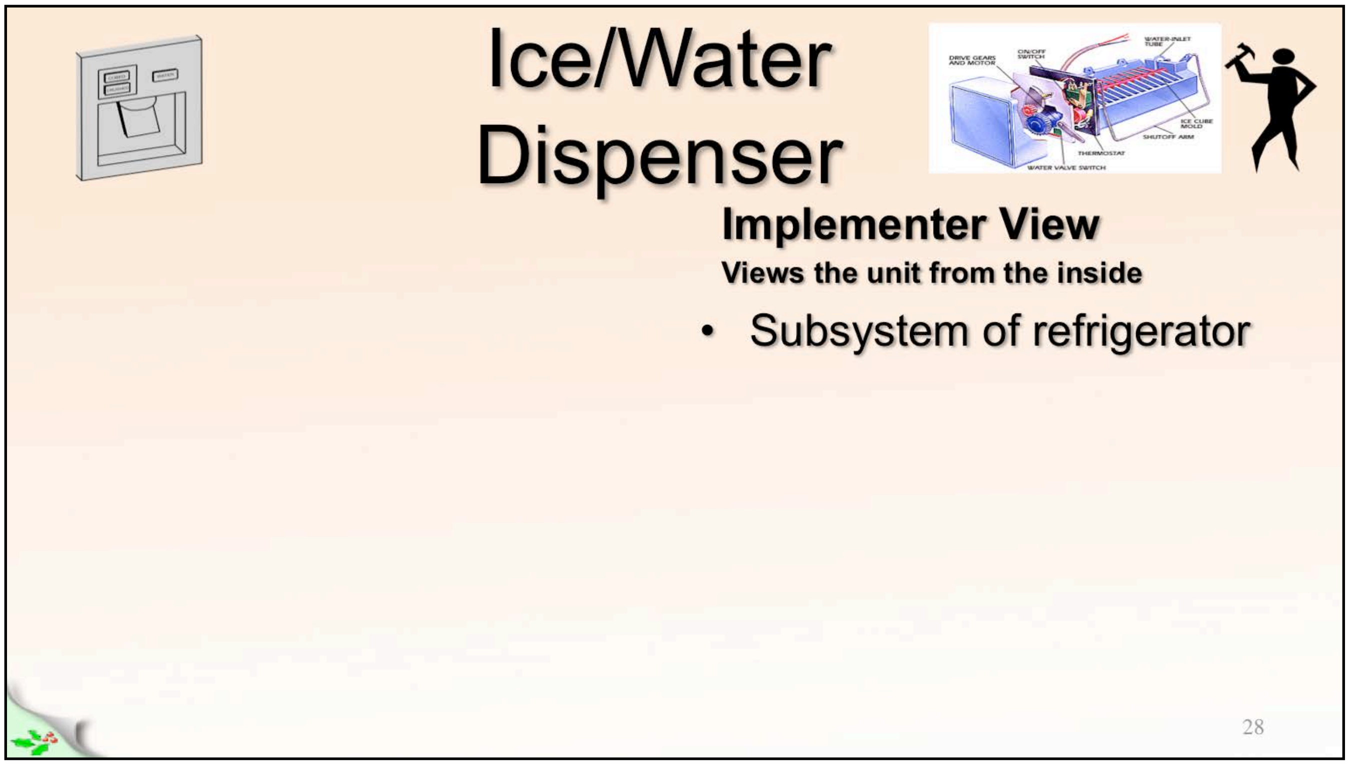
Ice/Water Dispenser



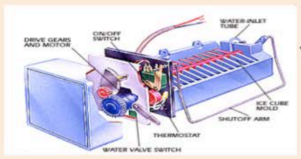
Implementer View

Views the unit from the inside

1. Now let’s now switch to the implementer’s view, and take a look at the system from the inside.



Ice/Water Dispenser

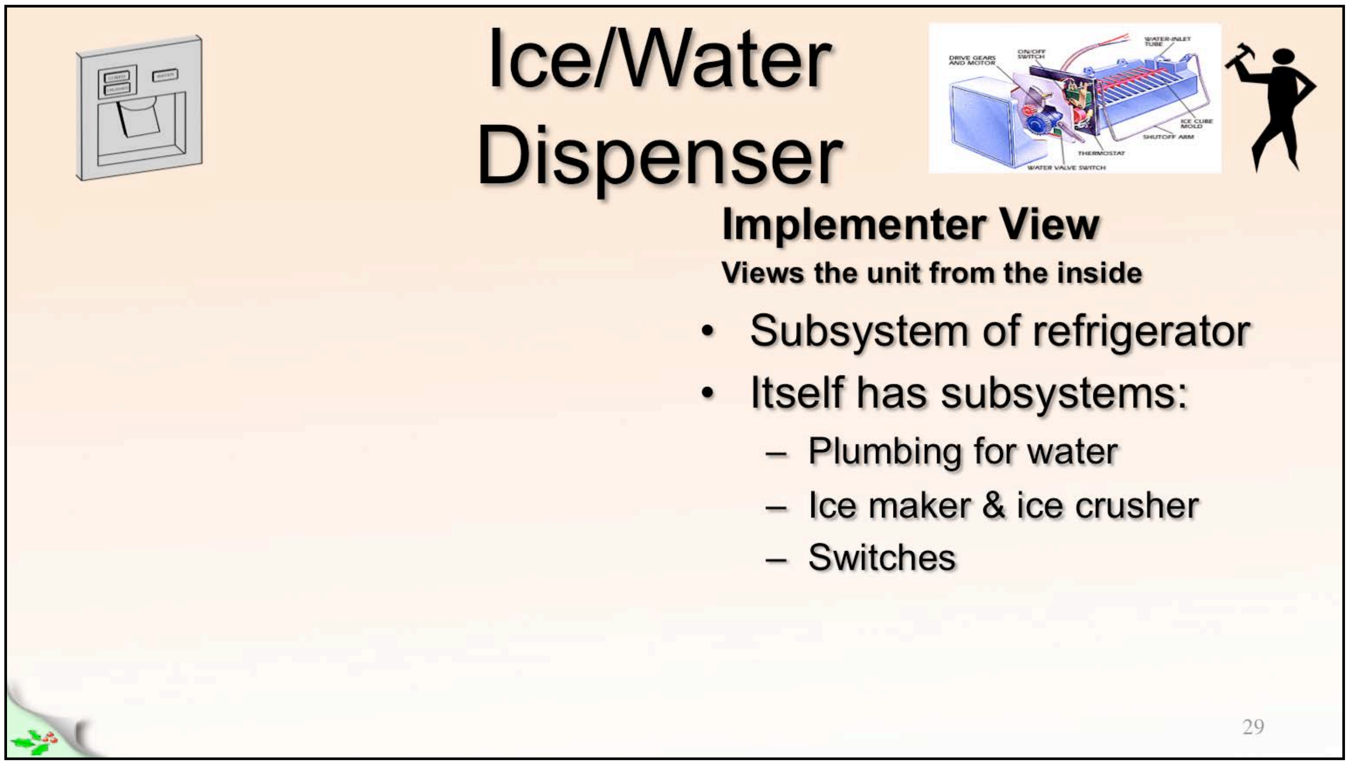


Implementer View

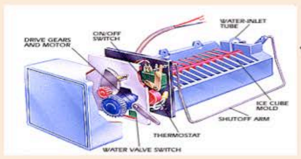
Views the unit from the inside

- Subsystem of refrigerator

1. First, we must recognize that the Ice & Water Dispenser is itself a subsystem, a component of a larger system, the refrigerator.



Ice/Water Dispenser

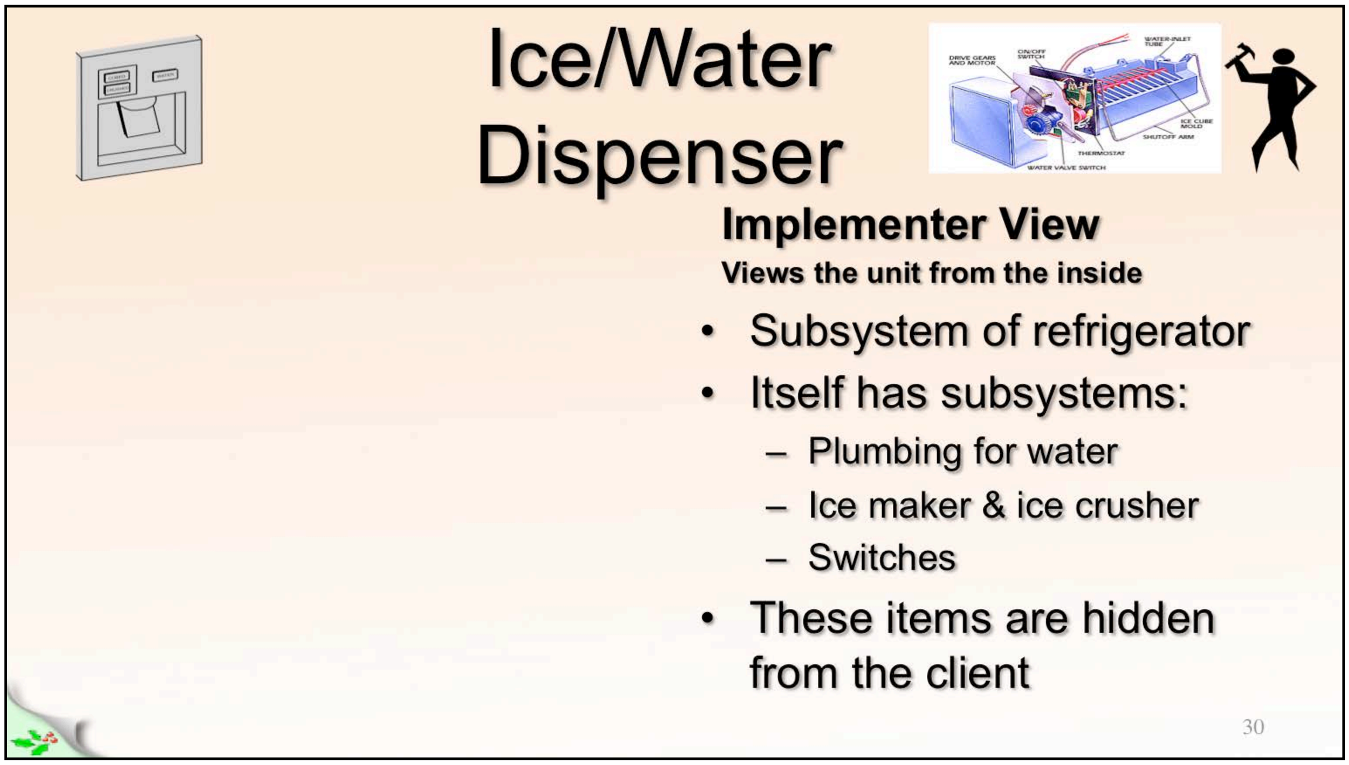


Implementer View

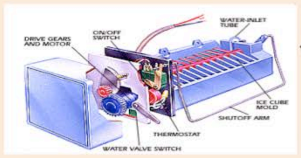
Views the unit from the inside

- Subsystem of refrigerator
- Itself has subsystems:
  - Plumbing for water
  - Ice maker & ice crusher
  - Switches

1. Now, diving inside of the Ice & Water Dispenser, we see that it is made up of smaller components – for example, the plumbing, the ice maker, the machine that crushes the ice, the electrical subsystem, and so on.
2. All of these internal components are hidden from the client of the Ice & Water Dispenser.



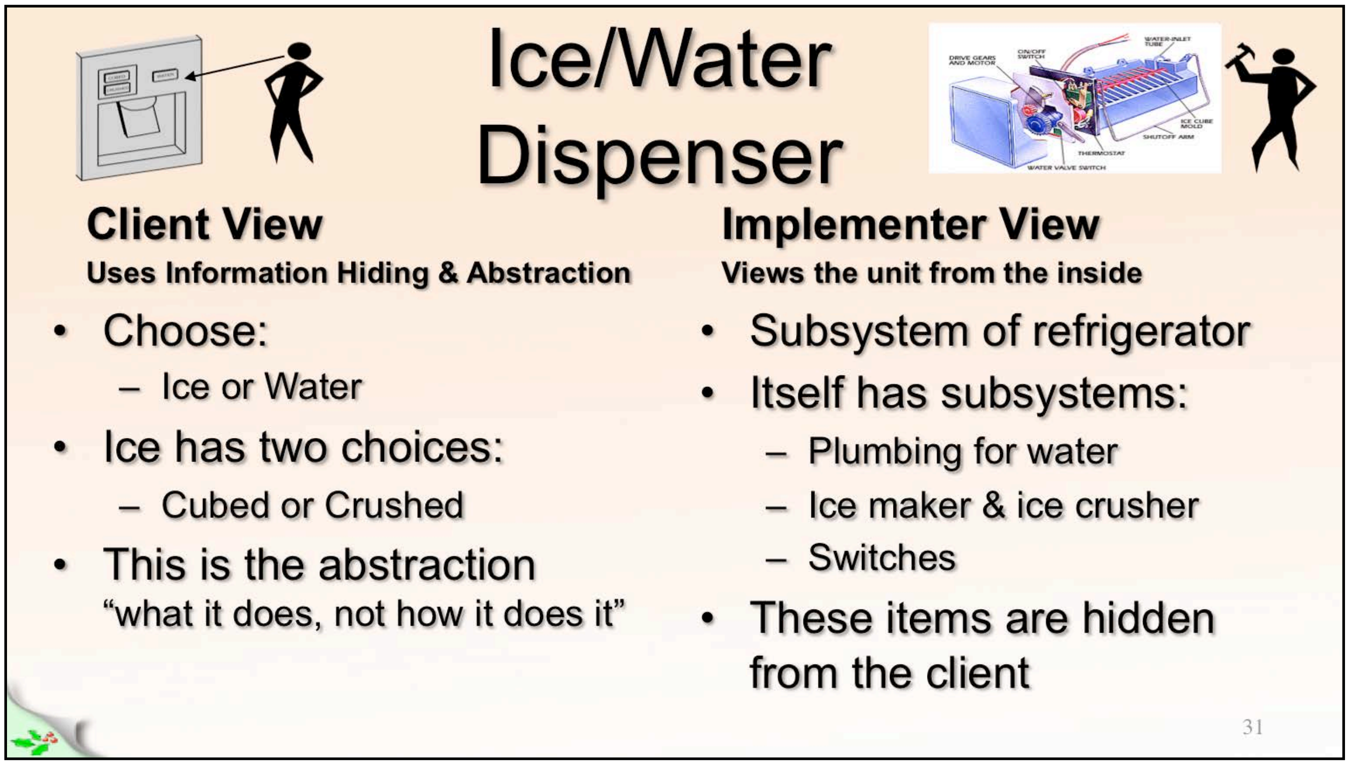
Ice/Water Dispenser

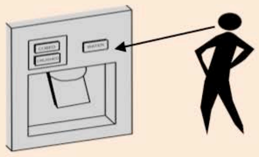


Implementer View

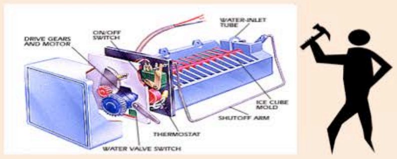
Views the unit from the inside

- Subsystem of refrigerator
- Itself has subsystems:
  - Plumbing for water
  - Ice maker & ice crusher
  - Switches
- These items are hidden from the client





Ice/Water Dispenser



Client View

Uses Information Hiding & Abstraction

- Choose:
  - Ice or Water
- Ice has two choices:
  - Cubed or Crushed
- This is the abstraction
  "what it does, not how it does it"

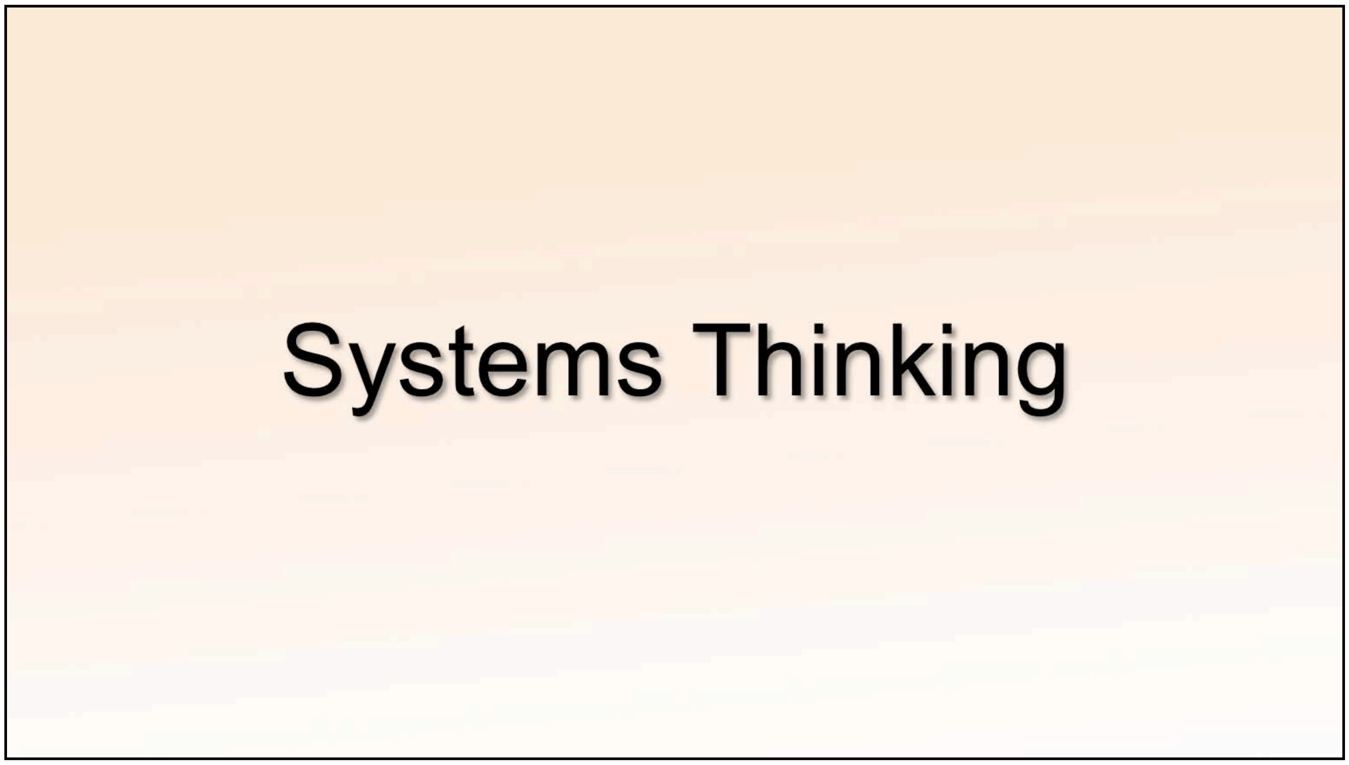
Implementer View

Views the unit from the inside

- Subsystem of refrigerator
- Itself has subsystems:
  - Plumbing for water
  - Ice maker & ice crusher
  - Switches
- These items are hidden from the client

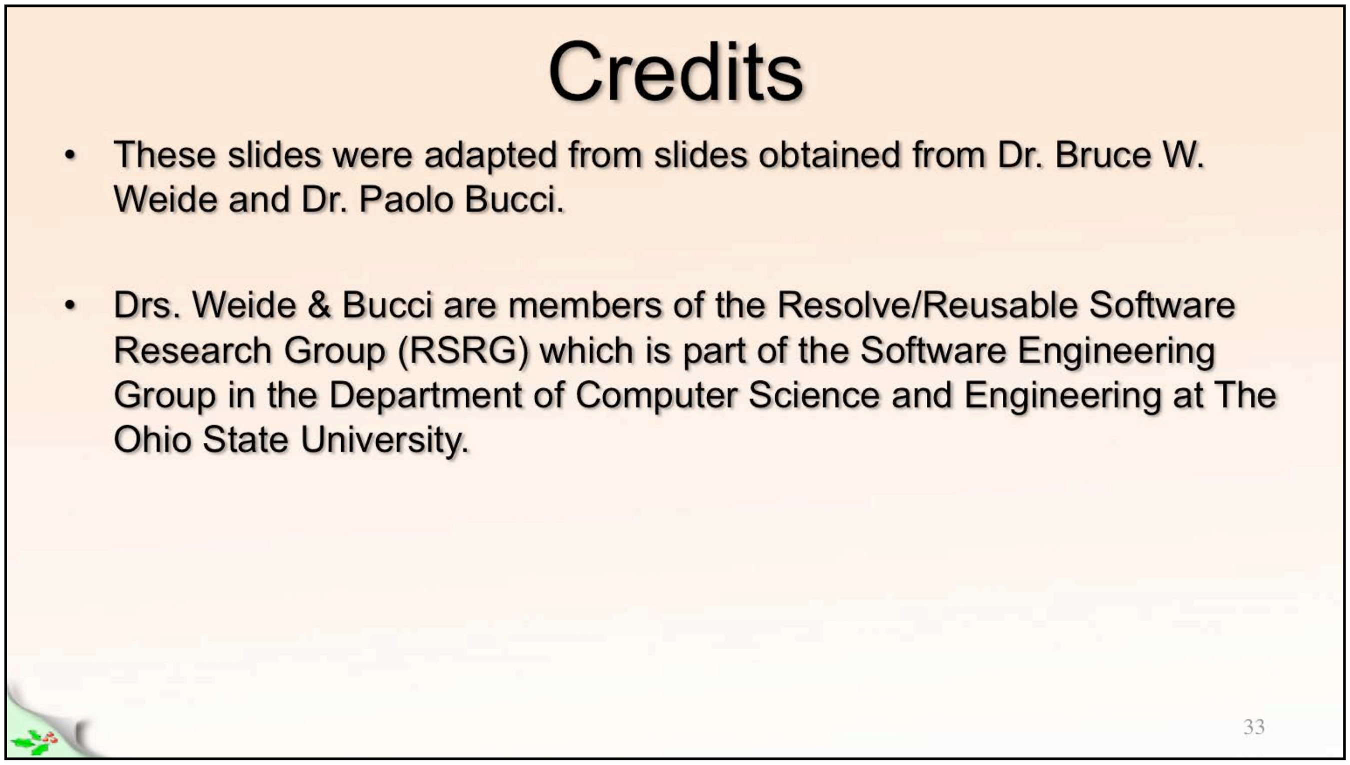
31

1. Finally, this slide shows both the client and the implementer view
2. The client thinks about *what* the Dispenser does by using the abstraction.
3. The implementer uses the abstraction to determine what internal components are needed in order to make the Dispenser work the way it is supposed to.



Systems Thinking

1. And there you have it, Systems Thinking.
2. When we cover future topics, we will use the terms and concepts discussed today.
3. One topic in particular is called Design by Contract – where we more fully develop the idea of the interface, using pre & post conditions.
4. We'll use those pre & post conditions to better outline the responsibilities of the client and the implementer.
5. Until next time –we'll see you again with more Notes on Engineering Software.



Credits

- These slides were adapted from slides obtained from Dr. Bruce W. Weide and Dr. Paolo Bucci.
- Drs. Weide & Bucci are members of the Resolve/Reusable Software Research Group (RSRG) which is part of the Software Engineering Group in the Department of Computer Science and Engineering at The Ohio State University.

33

1. These slides were adapted from slides obtained from Dr. Bruce W. Weide and Dr. Paolo Bucci.
2. Drs. Weide & Bucci are members of the Resolve/Reusable Software Research Group (RSRG), which is part of the Software Engineering Group in the Department of Computer Science and Engineering at The Ohio State University.