Human Computer Interaction

Leftovers: Adapters + Glue + Needfinding

Prof. Andrew D. Bagdanov

Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Firenze
andrew.bagdanov AT unifi.it

November 21, 2017

Outline

Overview

Overview

Today

- There are a number of topics that have become backed-up and/or delayed.
- I will talk about: adapters, glue technologies, and needfinding.

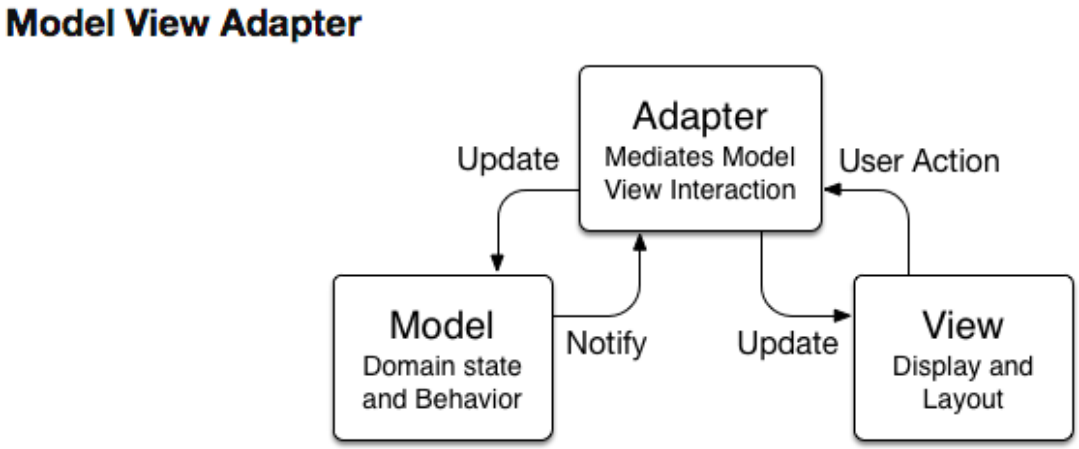
Rest of the week

- The laboratory for this week IS MOVED TO FRIDAY.
- Tomorrow: More needfinding.
- Friday: Laboratory (adapters, programming assignments).

Adapters

Adapters

- In the lecture on the MVC paradigm I asked you to investigate the Kivy Adapter classes.
- There is a recurring problem when implementing user interfaces using the MVC pattern.
- The Adapter (sometimes called Model-View-Adapter, or Mediating-controller MVC) architectural pattern is often encountered when we want to present large amounts of data to the user.
- How can you implement View classes so as to minimize the knowledge the Views need to have about the underlying Model?



Adapters

- If the View needs to understand all of the underlying logic of the Model, it was hardly worth separating them in the first place...
- In the other direction, if in our Model we have to implement an API specifically to create View-specific components (e.g. Labels and Buttons), again we are mixing Presentation and Data logic.
- These problems become especially noticeable when dealing with complex View classes, like TextLists or Trees.
- Goal: maintain good separation of concerns, but avoid *ad hoc* controller implementation as much as possible.
- Why?

Adapters

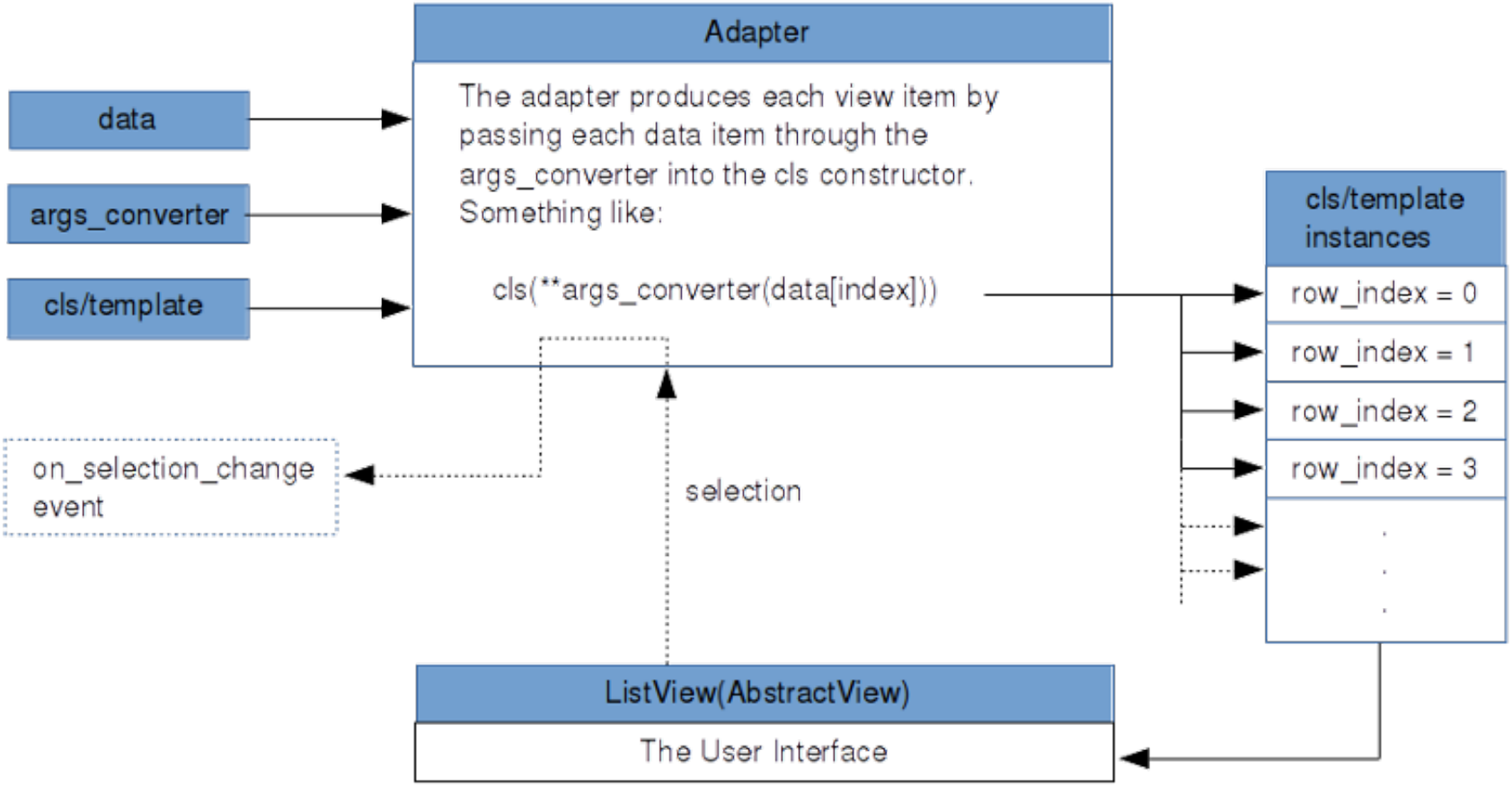
- A solution to these problems that most advanced GUI frameworks offer is a system of Adapters.
- An Adapter, at a general level, is simply a class that we can use to adapt the Model data, to the data model required by a specific View.
- For example, there might be a TreeItemAdapter class (it will have pure virtual methods in it) that we can extend.
- Our extension will interface our Model with what the TreeView expects its Items (e.g. internal nodes and leaves in the tree) to look like (in terms of API).
- This doesn't entirely solve the mixing presentation and data, but it isolates it in the Adapter and we can leave our Model and Views pure.

The Kivy Adapter model

- Kivy supports this Adapter concept for specific, top-level widgets.
- A Kivy Adapter is a mediating controller-type class that processes and presents data for use in views.
- It does this by generating models, generally lists of SelectableView items, that are consumed and presented by views.
- You can think of an adapter as a sort of model converter.
- Views in Kivy (as far as Adapters are concerned) are top-level widgets, such as a ListView or a TreeView.

The Kivy Adapter model

- This diagram is useful for understanding the relationship between Models and Views via Adapters:



The Kivy Adapter model

The components involved in this process are:

- Adapters: mediate between the user interface and your data.
  - Manages creation of view elements using the `args_converter` to prepare the contructor arguments for your cls/template view items.
  - Adapter is subclassed by `SimpleListAdapter` and `ListAdapter`.
  - `DictAdapter` is a more advanced and flexible subclass of `ListAdapter`.
- Models: data for which an adapter serves as a bridge to views.
  - Can be any sort of data.
  - Model mixin classes can ease the preparation or shaping of data for use in the system.
  - For selection operations, `SelectableDataItem` can optionally prepare data items to provide and receive selection information.

The Kivy Adapter model

- Args Converters: written by you to convert data items.
  - Convert data to dictionaries suitable for instantiating views.
  - They take a row of your data and create dictionaries passed into the constructors of your cls/template.
- Views: models of your data presented to the user.
  - Each data item creates a corresponding view subitem (the cls or template).
  - Presented in a list (or tree, or whatever) by the View.

The Kivy Adapter model

- What should we take away from this?
- We need to implement an Adapter to map from our Model to something called a "cls/template" in Kivy.
- This Adapter will need to have an Args Converter supplied to it that specifies how to do this mapping.
- Our base model can be a List or a Dict.
- Let's build a simple example using a ListAdapter to convert a list of CounterButtons into something a Kivy ListView can display.

Example: ListView item adapter

- Let's look at some examples using components we already have.
- This was the code to load our CounterButton widget views from the KV source:

```
class ButtonWithModel(Button):
    def __init__(self, model, **kwargs):
        self.model = model
        super(Button, self).__init__(**kwargs)

# Explicitly load the KV file (applies rules).
Builder.load_file('./counterbutton.kv')
```

- Now we will add a new element to our Model: a list of CounterModels

```
# Now a new component of our model: a list of CounterModels
buttons = [CounterModel() for i in range(100)]
```

- This should properly be put in the Model package, but let's pretend.
- Our goal: adapt this model for use in the Kivy ListView widget.

Example: ListView item adapter

- We will use the `SimpleListAdapter` class to do this.
- The `data` of the adapter is precisely our new Model element: a List of CounterModels.
- The `cls` argument of the adapter is the constructor of the Widget that will be generated.
- We must then provide an `args_converter` function that takes a `(row_index, data_item)` pair and generates the argument list for the widget specified in `cls`.
- Whenever the `ListView` needs the list items, it will call the supplied `args_converter` and `cls` constructor.

```
# First stab, use the default ListConverter.
def first():
    list_adapter = SimpleListAdapter(
        data=buttons,
        args_converter=lambda row, model: {'model': model,
                                           'size_hint_y': None,
                                           'height': 50},
        cls=Factory.CounterIntervalButton)
    return ListView(adapter=list_adapter)
```

Example: ListView item adapter

- OK, that's interesting, but what if we want to do something more complex?
- For example, what if we want a composite widget (i.e. a `CounterButton` and a `ResetButton` alongside it).
- Well, Kivy provides the `CompositeListItem` precisely for this purpose:

```
args_converter = lambda row_index, rec: \
        {'text': rec['text'],
         'size_hint_y': None,
         'height': 25,
         'cls_dicts': [{'cls': ListItemButton,
                        'kwargs': {'text': rec['text']}},
                       {'cls': ListItemLabel,
                        'kwargs': {'text': "Middle-{0}".format(rec['text'])
                                   'is_representing_cls': True}},
                       {'cls': ListItemButton,
                        'kwargs': {'text': rec['text']}}]}
```

Example: ListView item adapter

- OK! So, let's put this CompositeListItem together!
- [45 minutes of cursing in every language I know]
- The `CompositeListItem` class in Kivy seems hopelessly broken: it makes assumptions about the classes it contains that are undocumented.
- Then, of course, there is this:

Adapter

Module: kivy.adapters.adapter

> Warning
> This code is still experimental, and its API is subject to change in a future version.

- Well, crap.
- Can we roll our own? How would we go about doing this?

Example: ListView item adapter

- Well, conceptually the widget we want in the list is just a horizontal BoxLayout that holds a CounterButton and another regular Button.
- Maybe something like:

```
class CounterListItem(BoxLayout):
    def __init__(self, model, index, **kwargs):
        super(CounterListItem, self).__init__(**kwargs)
        self.orientation = 'horizontal'
        self.add_widget(Factory.CounterIntervalButton(model))
        self.add_widget(Button(text='Reset'))

def second():
    list_adapter = SimpleListAdapter(
        data=buttons,
        args_converter=lambda row, model: {'size_hint_y': None,
                                           'height': 50,
                                           'model': model},
        cls=CounterListItem)
    return ListView(adapter=list_adapter)
```

Example: ListView item adapter

- The Kivy Adapter functionality does help link Model with View, but the API is not very well-documented.
- With some experimentation, though, it is pretty easy to figure out what is going on.
- In this example, we also could have specified our `CounterListItem` in the KV design language.
- All Adapters can take a `template` argument instead of `cls`, which specifies a string containing the KV definition of your widget.
- We will see more examples of this in the laboratory on Friday.

Glue technologies

Glue technologies

- Much of the work that we do (in computer science in general, but also specifically in HCI) involves putting together a variety of disparate technologies.
- A system might use a Kinect sensor, and connect to an interactive table or wall-scale display.
- Invariably, there is a need to link together these system in some coherent way.
- This is where glue becomes useful.
- Glue technologies are libraries and protocols that allow us to create Middleware components that link things together.
- Today I want to talk briefly about two such technologies.

Glue technology: Open Sound Control (OSC)

- Open Sound Control (OSC) is "a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology."
- What does this have to do with HCI? Good question...
- When building HCI applications we often need to send information from one device to another, and this information needs to be of extremely high fidelity.
- Imagine sending multi-touch information from a remote touchscreen to an application. At what speed should this be sent?
- Even a superficial consideration of requirements should convince us that we need protocol and infrastructure support for this.
- And this need was first addressed in the sound engineering community (in fact, OSC was an early MIDI competitor).

Glue technology: Open Sound Control (OSC)

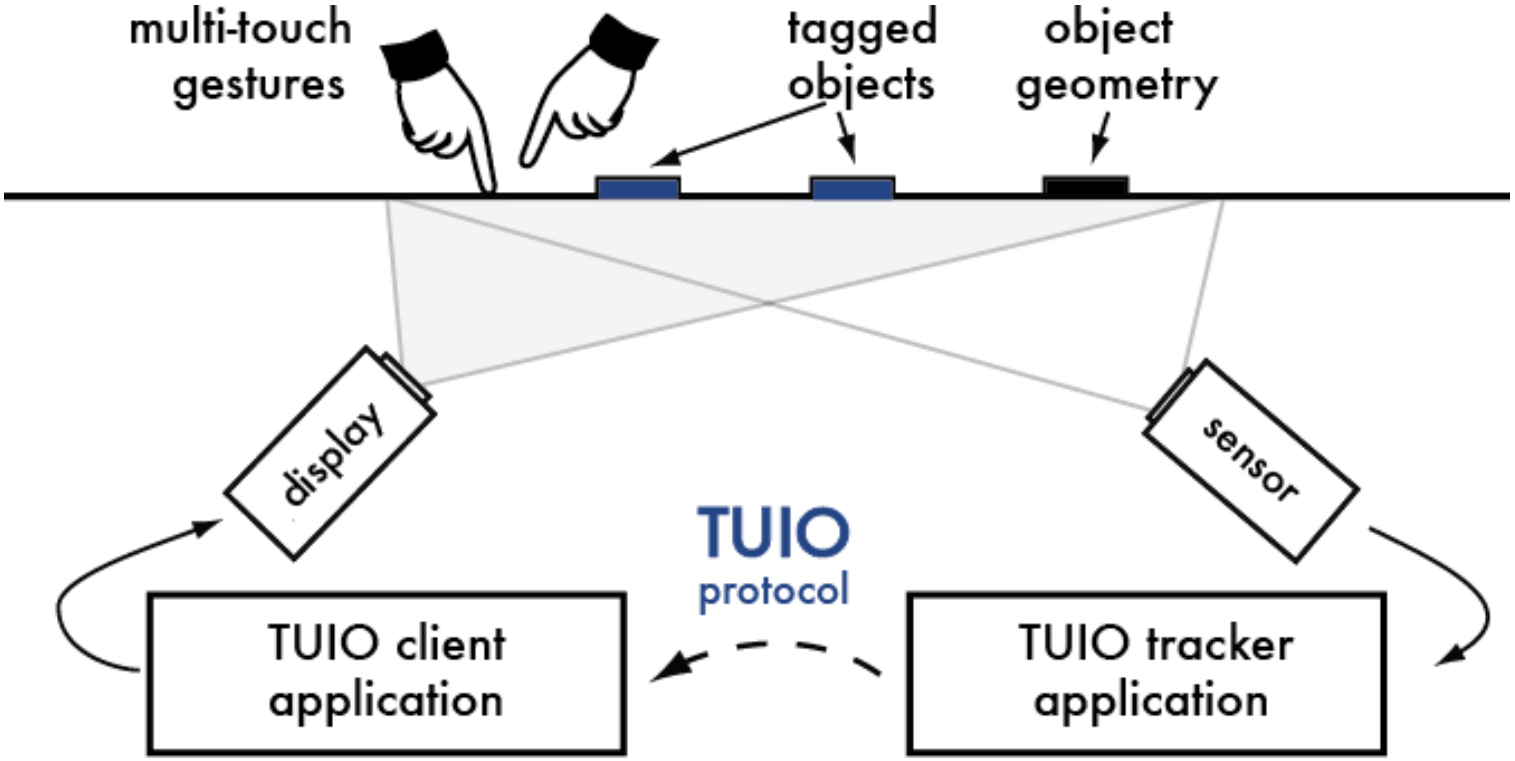
- OSC offers the following features:
  - Open-ended, dynamic, URL-style symbolic naming scheme
  - Symbolic and high-resolution numeric argument data
  - Pattern matching language to specify multiple recipients of a single message
  - High resolution time tags
  - "Bundles" of messages whose effects must occur simultaneously
  - Query system to dynamically find out the capabilities of an OSC server and get documentation
- This all sounds very fancy, but the OSC specification is really quite simple:
  http://opensoundcontrol.org/spec-1_0
- It is really a protocol for high-speed, high-fidelity inter-process communication.
- OSC supports synchronized, structured communication of data.
- Note: OSC is a low-level communications technology.

Glue technology: TUIO

- What if we want to send a very specific type of data between components?
- For example, what if we just need to send multi-touch events from one device (e.g. a touchscreen) and another (e.g. a GUI application)?
- Ideally we should be able to do this using a hardware-independent protocol to represent said multi-touch events (so the GUI is not inextricably linked to one device).
- And also ideally we shouldn't have to invent our own protocol for this, implementing it at great effort using OSC.
- Enter TUIO: building on top of OSC, TUIO is a multi-touch and tangible user interface protocol.

Glue technology: TUIO

- As always, a picture is worth a thousand words:



- TUIO is a client-server protocol: a server observes the multi-touch events, converts them to the TUIO multi-touch protocol, and broadcasts them; the client listens and reacts to these multi-touch events.

Glue technology: TUIO

- This type of architecture has many advantages.
- For example, we can separate the input processing (the tracker in TUIO terminology), from the GUI – they can be on different machines in different locations.
- TUIO message bundles look like this:

```
/tuio/2Dcur source application@address
/tuio/2Dcur alive s_id0 ... s_idN
/tuio/2Dcur set s_id x_pos y_pos x_vel y_vel m_accel
/tuio/2Dcur fseq f_id
```

- TUIO is a higher-level, domain-specific protocol on top of OSC.
- It is still fairly low-level, however.

Glue technology: TUIO

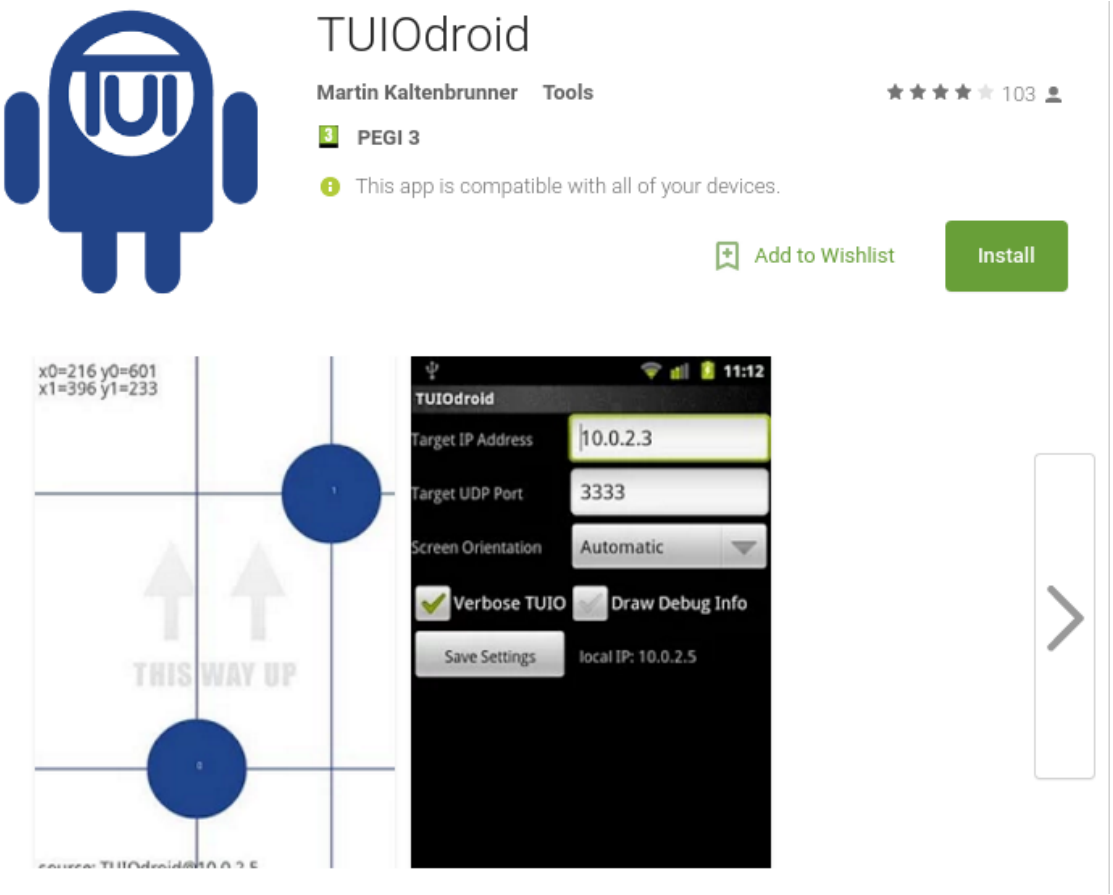
- Here is an interesting use-case:
  1. Write a Kivy application to do something.
  2. Use the Kivy TUIO input provider: https://kivy.org/docs/api-kivy.input.providers.tuio.html which can be attached to your application like this:

```
class TestApp(App):
    def build(self):
        Config.set('input', 'andy_tablet', 'tuio,192.168.1.2:3333')
```

This tells the application to use the TUIO stream from the given IP address and port as input to the application.

Glue technology: TUIO

3. Now, use this Android application to start up the TUIO stream:

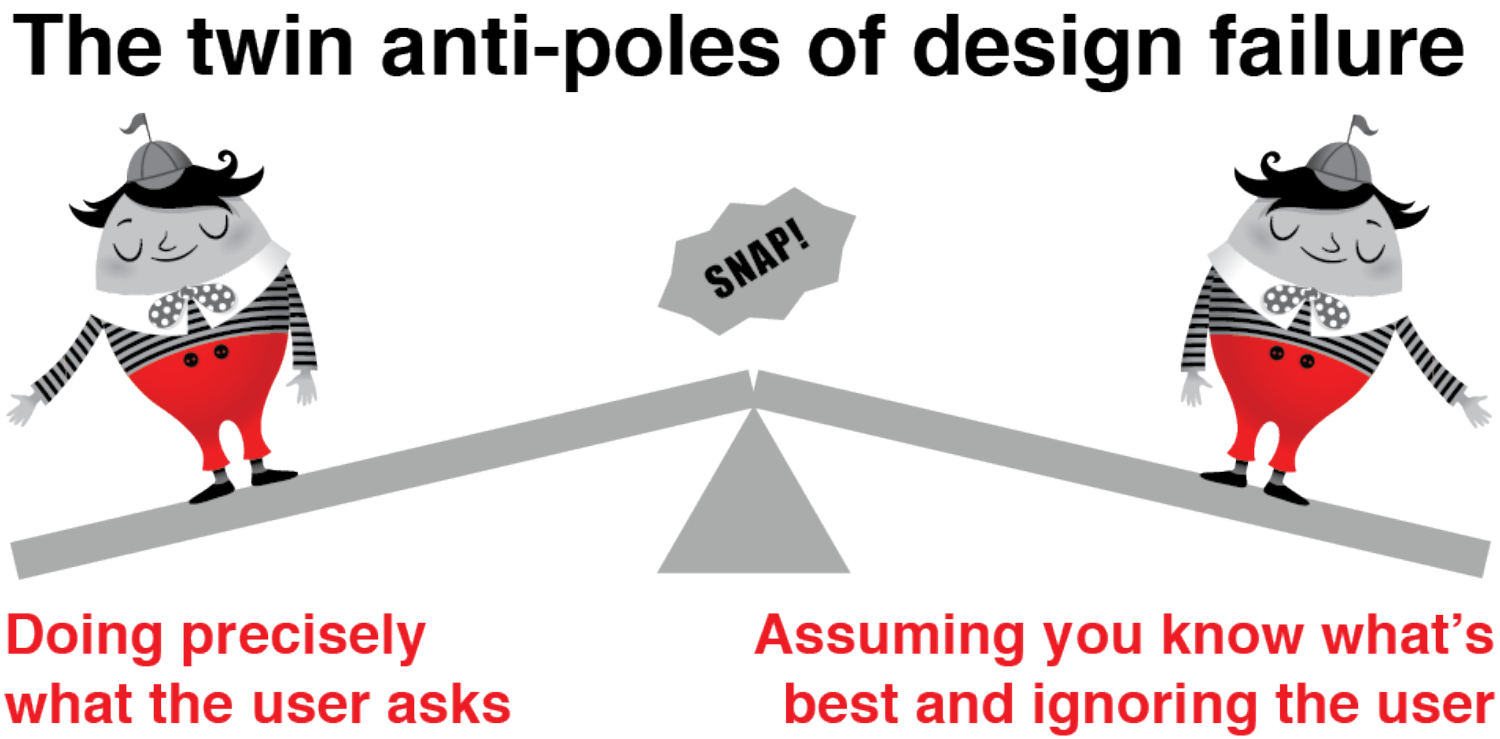


Needfinding

Needfinding

- Needfinding is the process of observing people to discover their needs, goals, and values.
- It is often associated with the process of developing new products or even new businesses.
- The main element is the investment of significant time, effort or money in the development of something new.
- Whether a new product or a new HCI system, it always makes sense to understand whether a genuine need exists.
- A good starting point is to clearly identify an existing problem or need, because finding a big problem and need often yields important untapped opportunities.
- Observing people also helps build empathy and think from their point of view.
- So, how do we observe people and identify their needs?

Needfinding (why?)



Needfinding: participant observation

- It is essential to observe the users and their behavior in context (performing the activity).
- This is vital to learning and understanding their experience.
- While observing, we seek answers to these questions:
  - What do people do now?
  - What values and goals do people have?
  - How are these particular activities embedded in a larger context?
  - What similarities and differences are there across people?
  - Are there any hacks or workarounds used?

Observations

Interviews

Extreme users, lead users

Needfinding: What do people do now?

- Getting a full understanding of the culture, practices, and rituals of your target audience provides an understanding of your audience that is extremely helpful.
- Digging deep into the motivations, emotions, and aspirations of your audience allows you to better understand where to begin.
- Take the UNIFI Forms example, it is essential to understand how real people manage this process today.
- This implies understanding how all types of users (i.e. professors, students, administrative staff, etc) do things.
- Too many software systems are implemented and deployed without a complete understanding of how things are currently managed.
- Without understanding this, it is hard to expect improvement over the *status quo*.

Needfinding: What values and goals?

- Most often, we want to build technologies that align with what people care about and what they hope to accomplish.
- This doesn't mean literally building what people have asked for: people often (usually) don't know how to achieve their goals – especially for disruptive technologies.
- Instead, we must design technologies that will weave themselves into the fabric of everyday life, even if they introduce new concepts and functionality.
- Main point: people cannot be relied upon to tell us how they should accomplish their goals, but through observation and interview we can uncover what those goals are, and what values should be preserved (e.g. saving time).

Needfinding: What is the context?

- For a public transportation user, a bus or subway segment is a part of a larger activity like getting to a friend's house, commuting to work, or going to the grocery store.
- By understanding the constraints and goals of the large activity, we can derive ideas that are otherwise missed if we think narrowly about the bus ride.
- By figuring out why someone would choose to take the bus or not take the bus, we as designers might end up with something more broad, like creating a mobile application that helps people figure out when a bus is coming when they need to.
- Taking this broader view can help us be more effective as designers by helping to design for the larger activity that people are engaged in.

Needfinding: Similarities and differences?

- In our bus example, a low mobility user might care about the accessibility of the bus, while somebody else may be concerned with the cost, and yet another with efficiency in getting to his destination.
- In the UNIFI Forms example, what similarities might there be between students and professors? Between professors and administrators?
- In many application contexts, there can also be tension between goals and values.
- It is important to understand what these commonalities and differences are in order to develop systems that are genuinely useful to everyone.

Needfinding: Hacks?

- Uncovering hacks that people have discovered for accomplishing tasks is a gold mine for designers.
- This is because hacks represent methods that accomplish actual goals and respect the values of people in the system.
- Often, these can be translated directly into features in our systems.

Needfinding: observing the experienced

- A good strategy for finding needs is to be an apprentice under someone who has experience with the area.
- One illustration of the power of being in the presence of an expert comes from Jack Whalen from Xerox PARC:
  - While studying a call center for photocopier repair, he found that diagnosing photocopiers over the phone is really hard.
  - Unsurprisingly, after studying the support staff they found that the most effective person was the person who had been working there the longest.
  - To their surprise, however, the second most effective person wasn't the person that had been working their the second longest, but the person who had been sitting next to the most effective person.
  - Thus, by sitting next to an expert, these repair technicians were able to pick up informal skills of doing repair work that aren't written down anywhere.

Needfinding: observing the experienced

- If you see something that catches your eye while being an apprentice, be sure to interact and validate it so you can better understand why things are done the way that they are.
- Additionally, pay attention to all the artifacts that compose people's work, because the ways that people have hacked their equipment to make their work more effective is an indication of ways to innovate.
- For example, Post-Its have traditionally been used as artifacts to help users more easily navigate different routines.
- A harmful attitude about designing, although less common nowadays, is the tendency to think people who can't figure out how to use technology are simply incompetent.

Needfinding: observing the experienced

- Using another example from Xerox PARC, Lucy Suchman recorded a video (that has now become legendary) of two people trying to produce a double sided copy of 50 pages of paper:
  - [VIDEO]
  - According to legend, when Lucy shared the video with the executives, they wrote the users off as dumb.
  - However, when it was revealed that the two users were Allen Newell and Ron Kaplan – two of Xerox's premier research scientists – they were no longer able to say that it was because the users did not know how to use technology.
  - Ultimately, this video shows that if you are unfamiliar with a particular piece of technology, it can be difficult to figure out how to use it without an intuitive user interface.

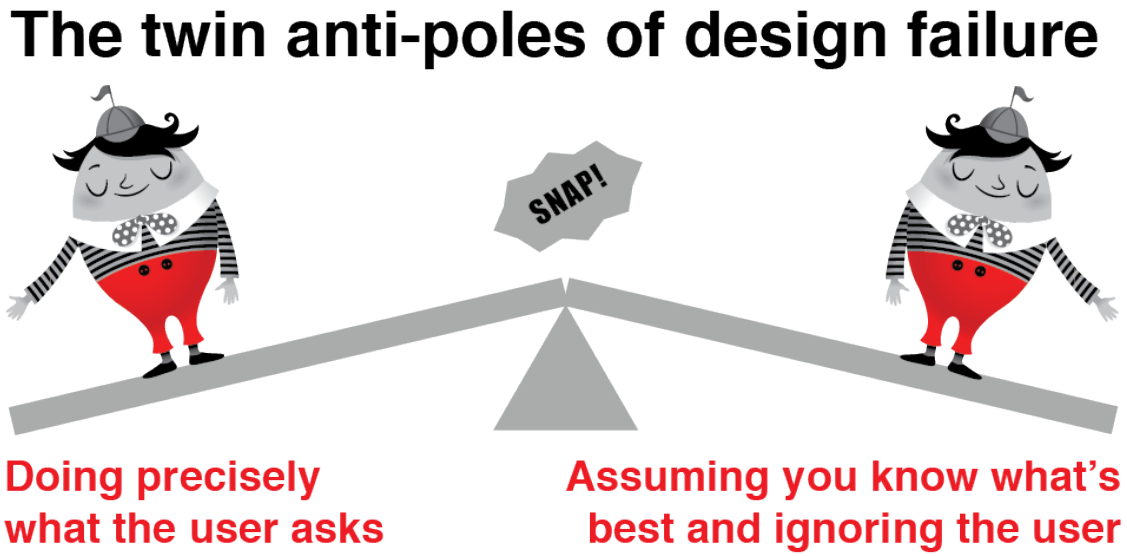
Needfinding: interviewing

- Of course, apprenticing with a company is a luxury for most people.
- Another way of observing people to identify needs is interviewing.
- When it comes to interviewing, one key distinction that must be made is the difference between what people say and what people do.
- For example:
  - Walmart conducted a study asking its customers whether they would like the aisles to be less cluttered.
  - Unsurprisingly, the participants of the study responded: well, yes.
  - Walmart then proceeded to declutter their aisles, remove inventory, and lost a billion dollars in sales.

Needfinding: interviewing

- What happened? In this situation, Walmart made two key mistakes:
  - First, they listened to what people said rather than paying attention to what people did.
  - Second, they asked a leading question.
  - Those two mistakes led them to do exactly the opposite of what would be most effective.

"If I asked people what they wanted, they would have said faster horses." – *Henry Ford*

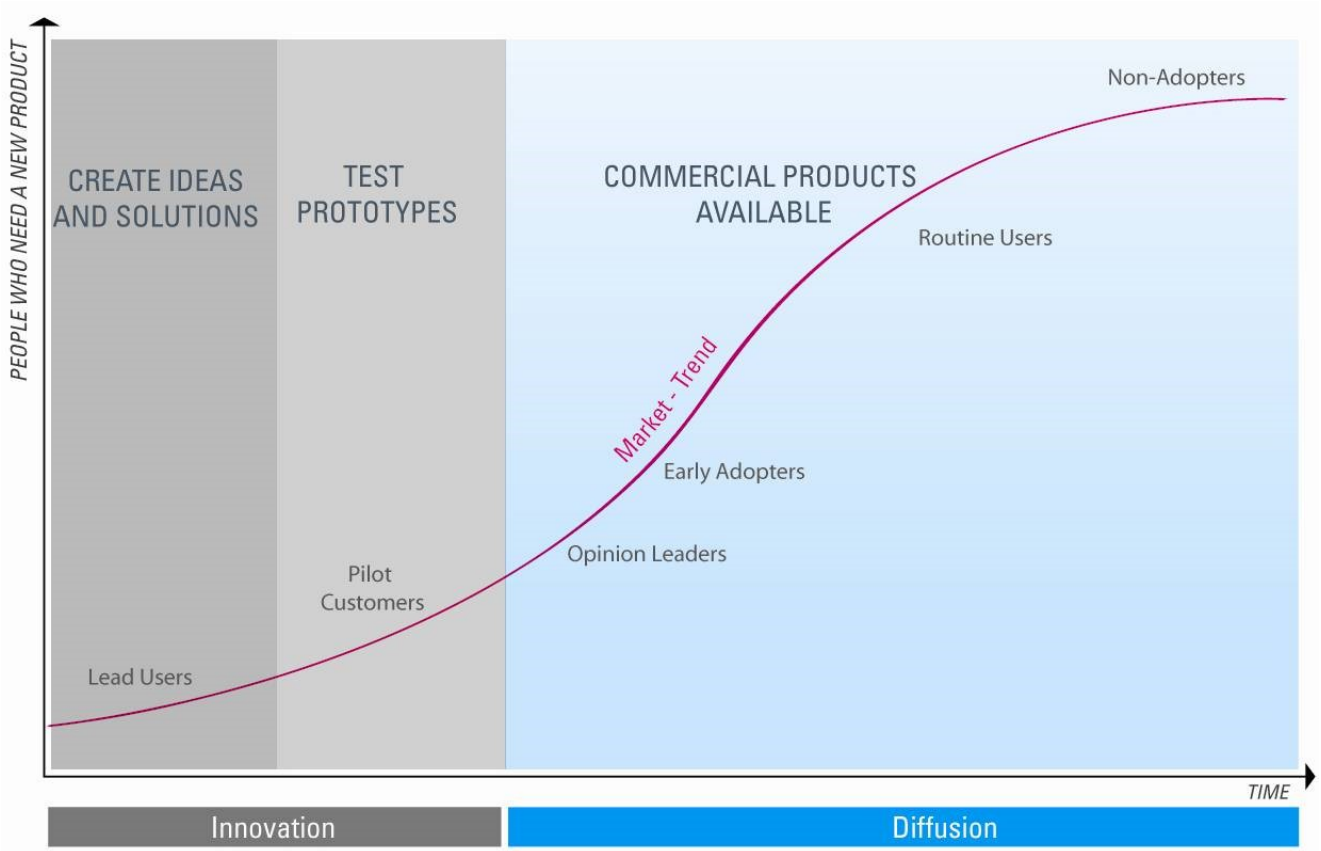


Needfinding: interviewing

- The starting point of any good interview is picking a good interviewee.
- A good collection of interviews from people of different backgrounds allows for diverse perspectives that can better help you spot trends that are common among different perspectives.
- If interviewing people for something you are building, you may be able to learn from people who are current users of similar systems.
- Good interviewees can be found through friends, family, and colleagues – however you may not always be able to get an interview with the ideal person.

Needfinding: interviewing

- The key is to be open to any insights that users or potential users may have.
- One group of users, known as lead users, can be extremely helpful in the interview process as well as the development process.
- Lead users are the knowledgeable early adopters that can help provide valuable feedback that directly informs design.



Needfinding: asking good questions

- What makes a good question when interviewing?
- Take this example and decide whether this is a good question:
  - The question is: "Is the daily update an important feature to you?"
  - When asked a question like this, most users would say "Yes."
  - It's kind of a leading question.
  - Most people, when asked if something's important, would say "Sure, why not?"
  - What's at stake? Why would you say no?
  - If you want to learn about the daily update, participant observation might be a lot more effective – you might even use log files to derive questions.
  - So, for example, you might ask somebody, "I see from the log that you've never used the daily update. Why is that? Tell me more."

Needfinding: asking good questions

- Other kinds of questions to avoid:
  - Questions that ask what users would do, like, or want in hypothetical situations. These questions often generate replies based on a person's ideal, hypothetical world, not necessarily what they would actually do.
  - Questions that ask how often users do things. Asking a user how often they go to the gym generates responses that reveal how often they wish they went to the gym, not actually how often they do.
  - Questions that ask how much they like things on an absolute scale. Asking a user how much they like something is often not extremely valuable or insightful.
  - Binary questions. Questions that only have two answer choices, like questions that ask on an absolute scale do not provide any significant insights into how you should build your product.

Needfinding: asking good questions

- Good questions are questions that are open-ended, have a specific goal, and are unbiased.
- For example, instead of asking how often a person goes to the gym, you should ask them to tell you about the most recent time that they went to the gym.
- Having a specific goal with each question allows you to uncover specific things with each question.
- If a question leaves your interviewee stumped for a little bit, that is a good thing.
- One common problem for new interviewers is that they often say that there's nothing to be found for the problem they're tackling because it's either impossible or obvious.

Needfinding: asking good questions

- However, it's rarely the case that there is nothing new. Malcolm Gladwell has a wonderful explanation in the introduction to his new book of collective stories "What the Dog Saw":

  "The trick to finding ideas is to convince yourself that everyone and everything has a story to tell. I say trick, but what I really mean is challenge, because it's a very hard thing to do. Shampoo doesn't seem interesting? Well, dammit, it must be, and if it isn't, I have to believe that it will ultimately lead me [to something] that is." Malcolm Gladwell

Needfinding: asking good questions

- If an interviewee's answer doesn't quite answer what you were trying to get out of the question, follow up with other questions that clarify what you're trying to get at.
- Paying attention to how your questions are answered also allows you to better understand how to ask good questions that generate the answers you're looking for.
- Examples of good questions:
  - When was the last time you used [some specific application]?
  - What were the first things you did when you woke up this morning?
  - What are three words you would use to describe your experience with [some specific task]?
  - What did you do after a [certain circumstance/situation] happened?

Needfinding: other interviewing aspects

- Another consideration to keep in mind is the location of the interview: they should take place in as realistic of a location as possible.
- For example, interviewing grocery shoppers is best done in grocery stores because it allows the interviewee to answer based on physical triggers that cause different behaviors.
- Conducting interviews in realistic location also allows you as the interviewer to better understand what your interviewee is talking about.

Needfinding: other interviewing aspects

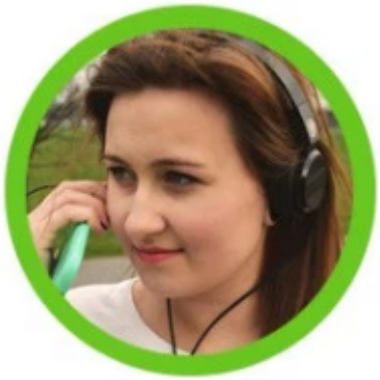
- A common question about interviewing is whether or not it should be recorded.
- Recording audio or video has benefits as well as drawbacks that should be considered.
- The drawbacks of recordings are that they can be time consuming to review and edit, requires permission, but most importantly it can change a participant's response.
- However, recordings provide a robust record and can help you focus on interviewing so you don't have to do that while taking notes.
- A good compromise is taking photos: they are quick and easy to take, provide a visual record, but does so without changing a participant's response.

Needfinding: personas

- The final step, after conducting as many user interviews as you possibly can, is to develop personas.
- Personas are fictional characters that you use to represent the demographics of your users.
- For example, if you are developing a messaging application, your personas can include teenage males who are using your messaging app to communicate with friends, as well as middle-aged mothers who are using the app to message their kids.

Needfinding: personas

- Example persona:



ZOE
age 18-22, single female, living with friends

Zoe studies as a graphic designer at a small art school. She aspires to one day work at an agency and eventually run her own.

She is constantly using dribbble to share her work and explore the work of others on her Macbook Pro. She regularly uses Photoshop and illustrator to create different designs, often showing her close friends before uploading it to dribbble.

She regularly uses a notebook to keep track of any sketches or ideas she may have regarding something that she would like to design.

Needfinding: personas

- The value in having personas is being able to quickly pinpoint different use cases among different demographics of people.
- By assigning a concrete name to a certain demographic, it becomes easier to keep that specific demographic in mind.
- Personas should be as detailed as possible, and be a compilation that represents different groups of your users.
- Beyond just demographic information, however, a persona should also capture a person's motivations, beliefs, intentions, behavior, and goals.
- In other words, give your persona a story to tell.

The way forward

The way forward

- In the next we will look at some concrete needfinding examples.
- Then, I will talk about how to take the needfinding process forward into the prototyping phase.
- Remember: tomorrow is a REGULAR LESSON.
- The laboratory is moved to FRIDAY.

Homework

Homework

Exercise 16.1: Zeroing in on a Project

During the week, spend some time thinking about a final project for the course. Try to at least determine:

1. Will it will be an individual or group project?
2. Which type of project: technical, standard, scientific, or some affine combination of the three?
3. What might be a source of users to select interviewees from?