6.033 Spring 2021

Lecture #1: Complexity, modularity, abstraction
plus an intro to client/server models
what is a system?
what is a system?

“a set of interconnected components that has an expected behavior observed at the interface with its environment.”
what is a system?

“a set of interconnected components that has an expected behavior observed at the interface with its environment.”

what makes building systems difficult?
what is a system?

“a set of interconnected components that has an expected behavior observed at the interface with its environment.”

what makes building systems difficult?

complexity
what is a system?

“a set of interconnected components that has an expected behavior observed at the interface with its environment.”

what makes building systems difficult?

complexity

why do we care?
what is a system?

“a set of interconnected components that has an expected behavior observed at the interface with its environment.”

what makes building systems difficult?

complexity

why do we care?

complexity limits what we can build
why do we care?

complexity limits what we can build
why do we care?
complexity limits what we can build

how do we mitigate complexity?
why do we care?

complexity limits what we can build

how do we mitigate complexity?

with design principles such as modularity and abstraction
why do we care?

complexity limits what we can build

how do we mitigate complexity?

with design principles such as modularity and abstraction

how do we enforce modularity?
why do we care?
   complexity limits what we can build

how do we mitigate complexity?
   with design principles such as modularity and abstraction

how do we enforce modularity?
   one way is to use a client/server model
why do we care?
complexity limits what we can build

how do we mitigate complexity?
with design principles such as modularity and abstraction

how do we enforce modularity?
one way is to use a client/server model

def main():
    html = browser_load_url(URL)
    ...
def server_load_url():
    ...
    return html

the browser is the client in this example
why do we care?

complexity limits what we can build

how do we mitigate complexity?

with design principles such as modularity and abstraction

how do we enforce modularity?

one way is to use a client/server model
why do we care?

complexity limits what we can build

how do we mitigate complexity?

with design principles such as modularity and abstraction

how do we enforce modularity?

one way is to use a client/server model

def main():
    html = browser_load_url(URL)
...

def server_load_url():
...
    return html

def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
why do we care?

complexity limits what we can build

how do we mitigate complexity?

with design principles such as modularity and abstraction

how do we enforce modularity?

one way is to use a client/server model

the browser is the client in this example

Class Browser
(on machine 1)

def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html

Class Server
(on machine 2)

def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html

def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```
```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html

def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```

Class Browser (on machine 1)

Class Server (on machine 2)
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html

def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
def main():
    html = browser_load_url(URL)
...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html

stub

def server_load_url():
    ...  
    return html

stub

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
stub
```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html
```

```
def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```
```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html
```
**def main():**
```
html = browser_load_url(URL)
...```

**def browser_load_url(url):**
```
msg = url  # could reformat
send request
wait for reply
html = reply  # could reformat
return html
```

**def server_load_url():**
```
...  
return html```

**def handle_server_load_url(url):**
```
wait for request
url = request
html = server_load_url(URL)
reply = html
send reply
```
```python
def main():
    html = browser_load_url(URL)
...

def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html

def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```

**Class Browser** (on machine 1)

**Class Server** (on machine 2)

load("view.html?item")
```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html
```

```
def server_load_url():
    ... return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```

**Client**

**Class Browser**
(on machine 1)

**Server**
(on machine 2)

**Network**

load("view.html?item")
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html

def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply

load("view.html?item")

stub

stub
def main():
  html = browser_load_url(URL)
  ...

def browser_load_url(url):
  msg = url  # could reformat
  send request
  wait for reply
  html = reply  # could reformat
  return html

def server_load_url():
  ...
  return html

def handle_server_load_url(url):
  wait for request
  url = request
  html = server_load_url(URL)
  reply = html
  send reply
```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html
```

```python
def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```
```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html

stub
```

```python
def server_load_url():
    ...
    return html

stub
```

```python
def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply

stub
```

load("buy.html?item&ccNo=XX")
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html

def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html

# stub

def server_load_url():
    ...  # stub
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html  # stub
    send reply
```

**Class Browser** (on machine 1)

**Class Server** (on machine 2)
Class Browser (on machine 1)

```python
def main():
    html = browser_load_url(URL)
...
```

```python
def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html
```

Class Server (on machine 2)

```python
def server_load_url():
    ...  
    return html
```

```python
def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```

Network flow:
- Client loads "buy.html?item&ccNo=XX"
- Server returns HTML
- Client loads "buy.html?item&ccNo=XX"
**client**

**Class Browser**
(on machine 1)

```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html
```

**server**

**Class Server**
(on machine 2)

```python
def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```

**network**

```python
load("buy.html?item&ccNo=XX")
```

**client**

**Network**

```python
load("buy.html?item&ccNo=XX")
```

**problem:** we just bought two copies of item
**Class Browser**
(on machine 1)

```python
def main():
    html = browser_load_url(URL)
...
```

```python
def browser_load_url(url):
    msg = url  # could reformat
    send_request
    wait for reply
    html = reply  # could reformat
    return html
```

**Class Server**
(on machine 2)

```python
def server_load_url():
    ...
    return html
```

```python
def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send_reply
```

**Network**

**Problem:** We just bought two copies of item

There are ways to deal with this issue — for example, giving each request a unique ID, and keeping track of those IDs on the server — but then new problems arise: for example, what happens if the server crashes in the middle of handling a request?
client  network  server
**scalability**: how does our system behave as we increase the number of machines, users, requests, data, etc.?
**scalability**: how does our system behave as we increase the number of machines, users, requests, data, etc.?
**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (☠️)? machines crashing, network links breaking, etc.
**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (☠)?: machines crashing, network links breaking, etc.
**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (☠)? machines crashing, network links breaking, etc.

**security:** how does our system cope in the face of targeted attacks ( зло)?
**scalability**: how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability**: how does our system deal with failures (☠) machines crashing, network links breaking, etc.

**security**: how does our system cope in the face of targeted attacks (😈)?
**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (☠)? machines crashing, network links breaking, etc.

**security:** how does our system cope in the face of targeted attacks (😈)?

**performance:** how do we define our performance requirements, and know if our system is meeting them? what do we do if performance is subpar (مادة)?
**scalability**: how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability**: how does our system deal with failures (☠)? machines crashing, network links breaking, etc.

**security**: how does our system cope in the face of targeted attacks (😈)?

**performance**: how do we define our performance requirements, and know if our system is meeting them? what do we do if performance is subpar (🐱)?

How do our design and implementation choices affect people and communities? Who makes those choices?
**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (☠)? machines crashing, network links breaking, etc.

**security:** how does our system cope in the face of targeted attacks (☠)?

**performance:** how do we define our performance requirements, and know if our system is meeting them? what do we do if performance is subpar ((#)?)

**how do our design and implementation choices affect people and communities? who makes those choices?**
http://mit.edu/6.033
http://mit.edu/6.033

has all of the class material, due dates, deadlines, etc.
http://mit.edu/6.033

has all of the class material, due dates, deadlines, etc.

Piazza
http://mit.edu/6.033

has all of the class material, due dates, deadlines, etc.

Piazza

this is where announcements happen. there will be things that we post on Piazza that aren’t on the website (e.g., zoom links)
complexity limits what we can build, but can be mitigated with design principles such as modularity and abstraction
complexity limits what we can build, but can be mitigated with design principles such as modularity and abstraction

you will see these principles applied over and over in this class

(a student once told me that I say “modularity” in almost every lecture, which seems correct)
complexity limits what we can build, but can be mitigated with design principles such as modularity and abstraction.

One way to enforce modularity is with a client/server model, where the two modules reside on different machines and communicate with RPCs; network/server failures are still an issue.

You will see these principles applied over and over in this class.

(a student once told me that I say “modularity” in almost every lecture, which seems correct)
complexity limits what we can build, but can be mitigated with design principles such as modularity and abstraction.

One way to enforce modularity is with a client/server model, where the two modules reside on different machines and communicate with RPCs; network/server failures are still an issue.

You will see these principles applied over and over in this class.

(a student once told me that I say “modularity” in almost every lecture, which seems correct)

Next lecture: naming, which allows modules to communicate.

After that: operating systems, which enforce modularity on a single machine.