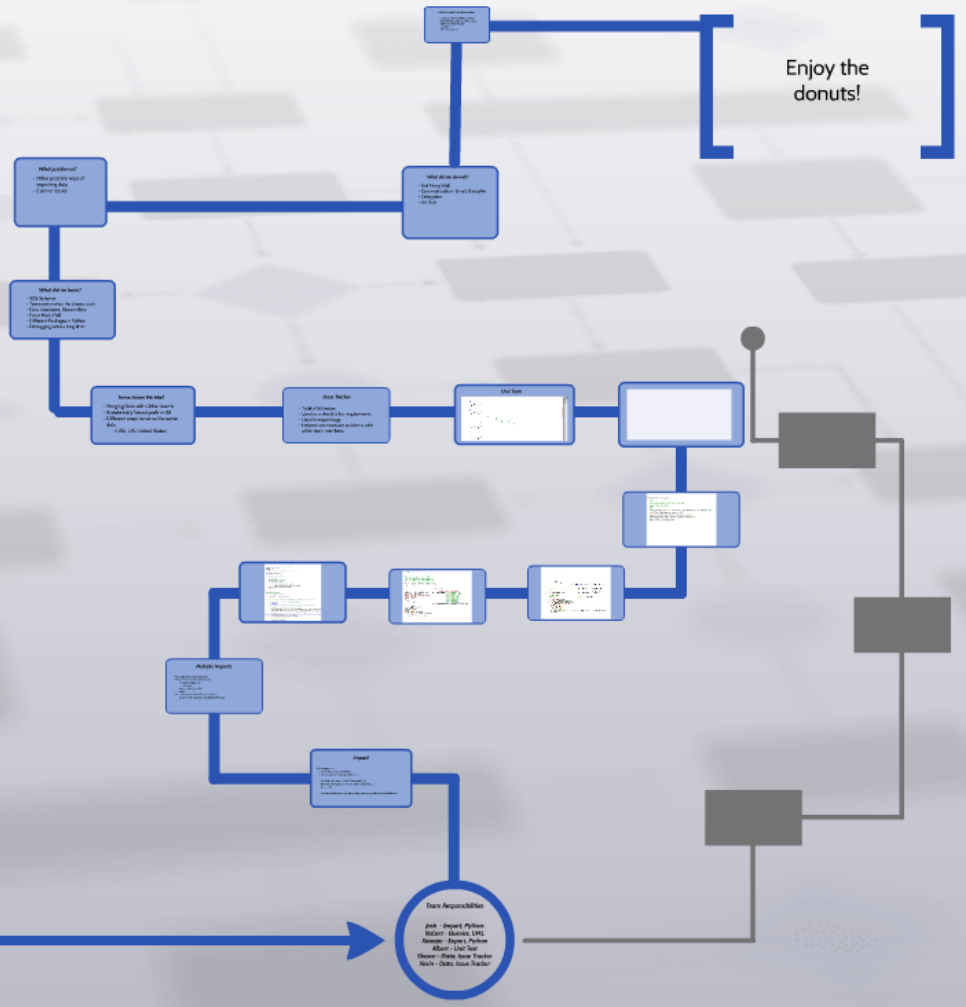
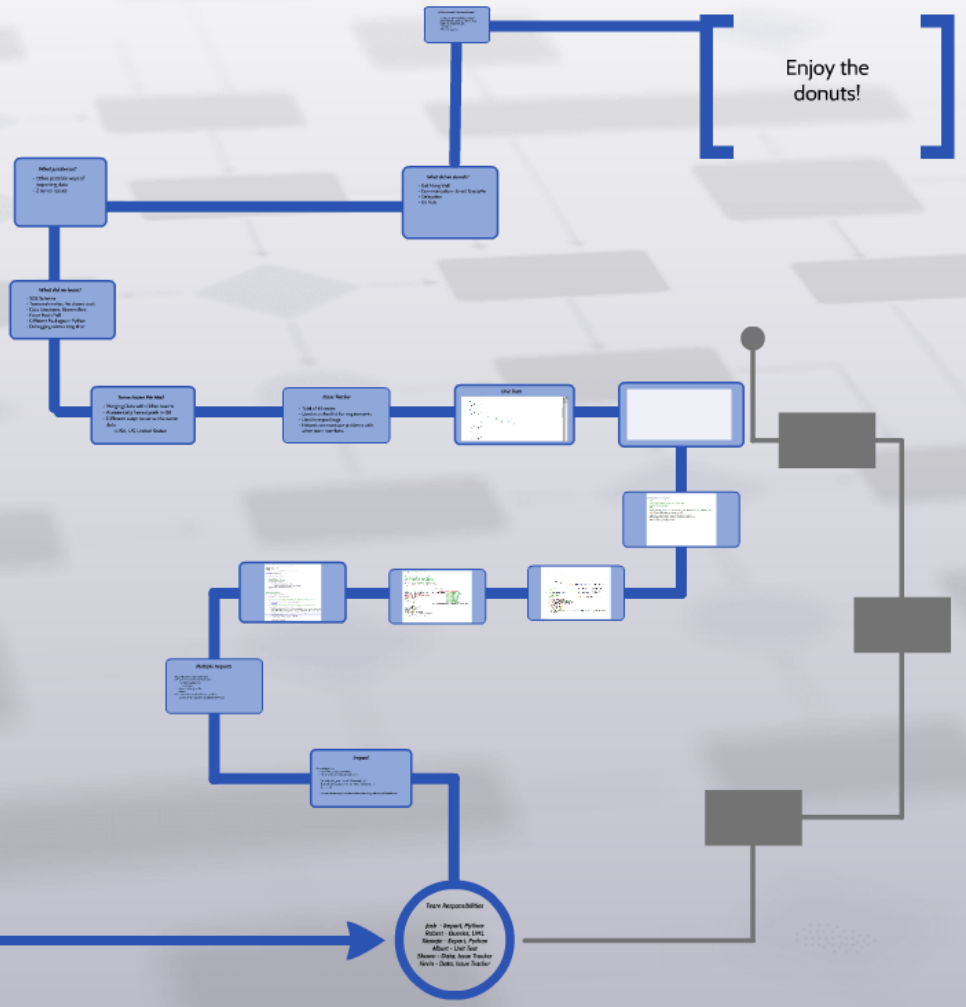


GottaGitThat

Josh Shen - Roberto Salinas - Kevin Boening
Albert Jin - Xiaoqin Li - Shawn Kulkarni



GottaGitThat

Josh Shen - Roberto Salinas - Kevin Boening
Albert Jin - Xiaoqin Li - Shawn Kulkarni

Team Responsibilities

Josh - Import, Python
Robert - Queries, UML
Xiaoqin - Export, Python
Albert - Unit Test
Shawn - Data, Issue Tracker
Kevin - Data, Issue Tracker

Import

Simple Approach

- Parse XML using ElementTree
- Put information in list of dictionaries

  [{'crisisId': CRI_001, 'name': 'Chernobyl', ...} \
  {'crisisId': CRI_002, 'name': 'UT Tower Shooting', ...}
  {..., ..., ...}]

- For every dictionary in list, turn values into string and query the database

Multiple Imports

- Read files from a list one by one
- Use global list to remove duplicates
  - If crisisId in global list:
    - continue
  - Append Id to global list
  - Query
- Use auto increment for Citations and Urls
  - Dictionary to map original value to new value

```
#------------------import people table-------------------------------
inserts_list = []
counter = 0
root_list = tree.findall("./people/person")
global people_id_list

for parent in root_list:
    insert_entry = {}
    #Iterates over Children
    for child in parent:
        if child.getchildren() == []:
            insert_entry[child.tag] = child.text
    inserts_list.append(insert_entry)
    counter += 1

#QueryInserting Loop
for i in range(0,counter):
    dict_entry = inserts_list[i]

    #--------------to check if current one is a duplicate----------
    if dict_entry['personId'] in people_id_list:
        continue
    #----------------------------------------------------------------
    people_id_list.append(dict_entry['personId'])
    s = (dict_entry.get('personId'), dict_entry.get('name'),dict_entry.get('
        dict_entry.get('streetAddress','Null'),dict_entry.get('city','Null')
        dict_entry.get('stateOrProvince','Null'),dict_entry.get('postalCode'
        dict_entry.get('country','Null'))
    s = 'insert into People Values' + str(s) + ';'
    s =s.replace('None', 'Null')
    t = wcdb_query(login,s)
```

```
def wcdb_solve(r,w,xml_filename_list):
    """
    r is a reader
    w is a writer
    login: Logs into DB, tree: Generates Element Tree,
    createDB(login): Creates Tables in DB,
    wcdb_import: import data from xml to databases
    wcdb_export: export data from databases to xml
    """
    a = ("z","joshen","pb6bKYnCDs","cs327e_joshen")
#     a = ("localhost", "root", "121314", "cs327e-wcdb")
    login_var = wcdb_login(*a)
    #-------------for acceptance tests-----------------
    createDB(login_var)
    r_flag = True
    data_tree = wcdb_read(r,r_flag)
    wcdb_import(login_var, data_tree)
    export_data = wcdb_export(login_var)
    wcdb_write (w, export_data)
    #----for real data from 8 groups------
    global crises_id_list
    crises_id_list = []
    global orgs_id_list
    orgs_id_list = []
    global people_id_list
    people_id_list = []
    global resources_id_list
    resources_id_list = []
    global waysToHelp_id_list
    waysToHelp_id_list = []
    createDB(login_var)
    r_flag = False
    for filename in xml_filename_list:
        data_tree = wcdb_read(r,r_flag,filename)
        wcdb_import(login_var, data_tree)
    export_data = wcdb_export(login_var)
    write = open('WCDB.out.xml', 'w')
    wcdb_write (write, export_data)
    write.close()


#--------Main fuction--------------------
def main():
    #---------- a list to save the filenames----------
    xml_filename_list = ['GottaGitThat-WCDB.xml',\
                         'UtNonOblivioscar-WCDB.xml',\
                         'SeekWolves-WCDB.xml',\
                         'BashKetchum-WCDB.xml',\
                         'TeamRocket-WCDB.xml',\
                         'Databosses-WCDB.xml',\
                         'EJADK-WCDB.xml',\
                         'Brigadeiros-WCDB.xml']
    WCDB.wcdb_solve(sys.stdin, sys.stdout,xml_filename_list)
    #---------------------------------------
```

```
def wcdb_export(login):
    """Generates ElementTree from DB
    root: <root></root>
    crises_tree: <crises></crises>
    """
    root = tree_builder('root')
    # ------------Crisis Export------------
    tree_counter = 0
    crises_tree = ET.Element('crises')
    root.append(crises_tree)

    crises = wcdb_query(
        login,
        """ select *
        from Crises;
        """)
    crises_tag_tuple = wcdb_query(
        login,
        """ show columns
        from Crises;
        """)
    for i in range(len(crises)):
        crisis_tree = ET.Element('crisis')
        root[tree_counter].append(crisis_tree)
        assert (type(crises[i]) is tuple)
        tag_counter = 0
        for entry in crises[i]:
            if entry == None:
                entry = 'NULL'
            root[tree_counter][i].append(tree_builder(crises_tag_tuple[tag_counter][0],entry))
            tag_counter += 1
```

```
def tree_builder(tag, content = ''):
    """builds 1 xml tree """
    builder = ET.TreeBuilder()
    builder.start(tag, {})
    builder.data(content)
    builder.end(tag)
    return builder.close()
```

```
def wcdb_write (w, data_tree):
    """
    converts an element string to a string data
    exports the string data
    """
    rough_exported_string = ET.tostring(data_tree, encoding='utf-8', method = "xml")
    assert(type(rough_exported_string) is str)
    reparsed = minidom.parseString(rough_exported_string)
    pretty_exported_string = reparsed.toprettyxml(indent="\t")
    w.write(pretty_exported_string)
```

Functions

createDB(login)
```
Create Needed Databases,
dropping if needed
```

tree_builder(tag, content='')
```
builds 1 xml tree
```

wcdb_export(login)
```
Generates ElementTree from DB
root: <root></root>
crises_tree: <crises></crises>
```

wcdb_import(login, tree)
```
Iterating through crisis tags in crises tag and import into DB: table Crises
```

wcdb_login(host, un, pw, database)
```
takes credentials and logs into DB
```

wcdb_query(login, s)
```
Logs into DB and runs provided string as query
```

wcdb_read(stdin, flag, filename=' ')
```
reads xml_filename_list
creates an element tree from string
flag is for reading choices
default filename is empty space
```

wcdb_solve(r, w, xml_filename_list)
```
r is a reader
w is a writer
login: Logs into DB, tree: Generates Element Tree,
createDB(login): Creates Tables in DB,
wcdb_import: import data from xml to databases
wcdb_export: export data from databases to xml
```

wcdb_write(w, data_tree)
```
converts an element string to a string data
exports the string data
```

Unit Tests

```
#--------------------------
# query function 3 in total
#--------------------------
def test_wcdb_query1(self):
  s = "drop table if exists CrisisOrgs"
  login = wcdb_login(*a)
  t = wcdb_query(login,s)
  self.assertTrue(t == None)

def test_wcdb_query2(self):
  s = "drop table if exists CrisisPeople"
  login = wcdb_login(*a)
  t = wcdb_query(login,s)
  s = """
        CREATE TABLE CrisisPeople (
        crisisId varchar(20) COLLATE utf8_unicode_ci NOT NULL,
        personId varchar(20) COLLATE utf8_unicode_ci NOT NULL,
        FOREIGN KEY (crisisId) REFERENCES Crises(crisisId),
        FOREIGN KEY (personId) REFERENCES People(personId)
        )ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
      """
  login = wcdb_login(*a)
  t = wcdb_query(login,s)
  self.assertTrue(t == None)

def test_wcdb_query3(self):
  s = """ select *
      from Crises;
      """
  login = wcdb_login(*a)
  t = wcdb_query(login,s)
  self.assertTrue(t != None)
```

Issue Tracker

- Total of 65 issues
- Used as a checklist for requirements
- Used to report bugs
- Helped communicate problems with other team members

Some Issues We Had

- Merging Data with Other teams
- Accidentally forced push in Git
- Different ways to name the same data

    -USA, US, United States

What did we learn?

- SQL Schema
- Teamwork makes the dream work
- Data Structures, ElementTree
- Force Push/Pull
- Different Packages in Python
- Debugging takes a long time

What puzzles us?

- Other possible ways of importing data
- Z server issues

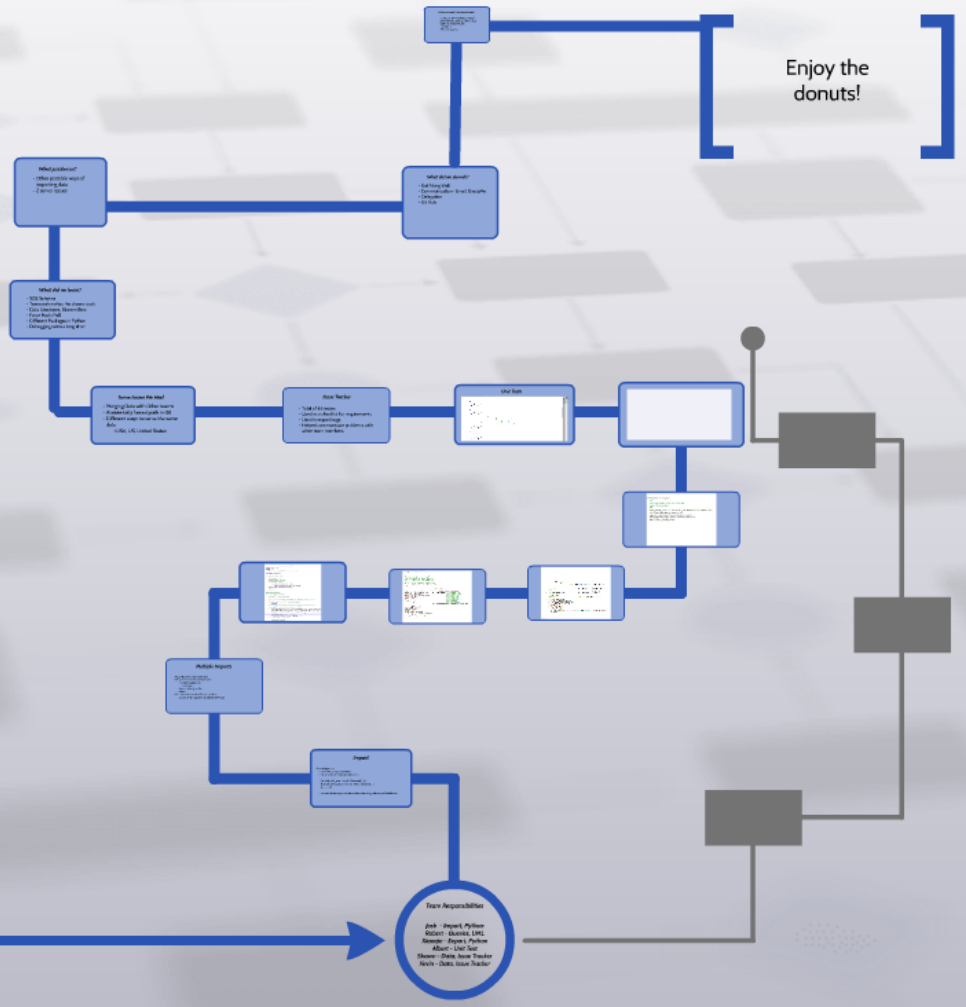
What did we do well?

- Got Along Well
- Communication- Email, GroupMe
- Delegation
- Git Hub

What we could have done better?

- Using SQL to fix duplicates instead
- Communicate better as a whole class
- Optimize Import/Export
  -Scalability
- Time Management

Enjoy the donuts!



GottaGitThat

Josh Shen - Roberto Salinas - Kevin Boening
Albert Jin - Xiaoqin Li - Shawn Kulkarni