University of
South Australia

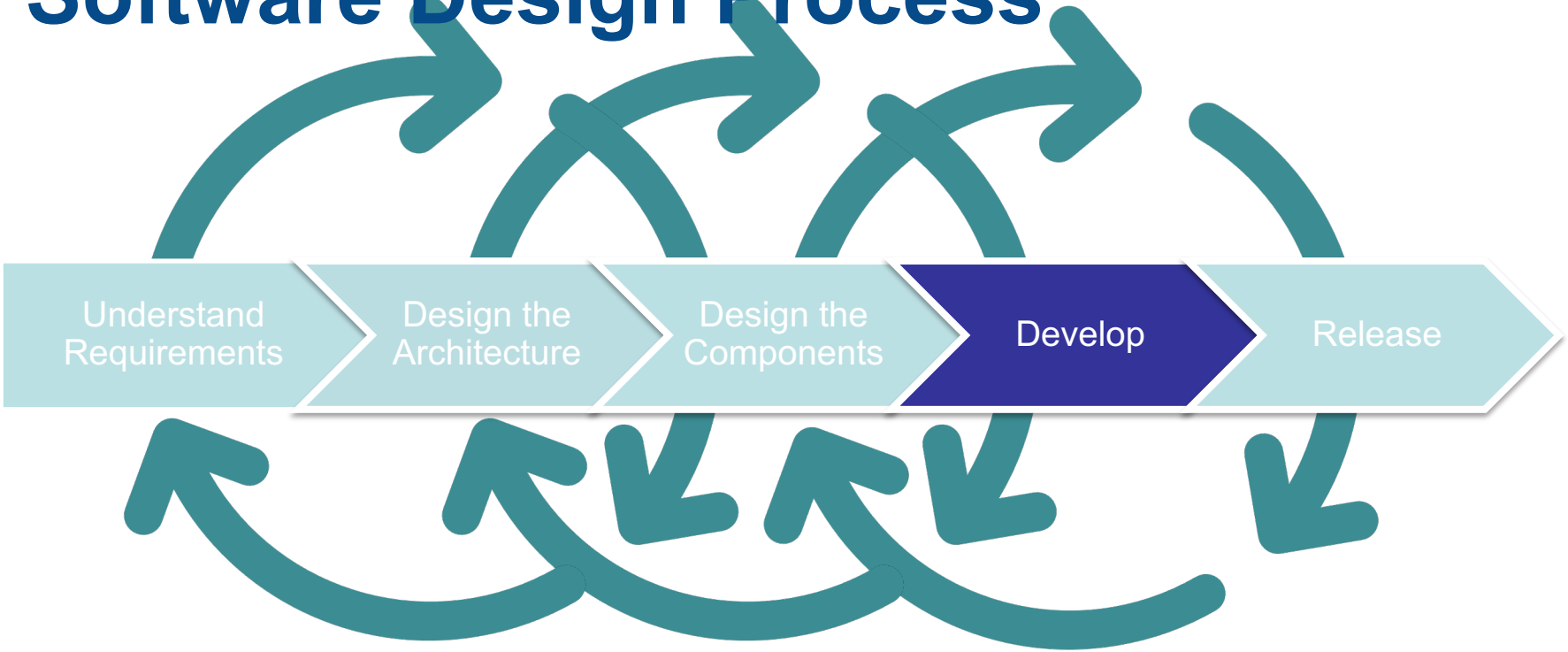
INFS 2044

Practical 2 Answers

Preparation

- Watch Unit Testing and Test Driven Development in Python on LinkedIn Learning
- Read the required readings
- Watch the Week 11 Lecture

Software Design Process



Learning Objectives

- Apply unit testing and test-driven development principles (CO4)

Task 1. Install pytest

- Install pytest

Open a command prompt (or terminal on MacOS/Linux) and run:

```
pip install --user pytest
```

Alternatively, follow the pytest install instructions for your platform/IDE.

Alternative Installation Instructions

To install:
C:\path\to\python.exe –m pip install pytest

To Run:
C:\path\to\python.exe –m pytest <file_containing_tests.py>

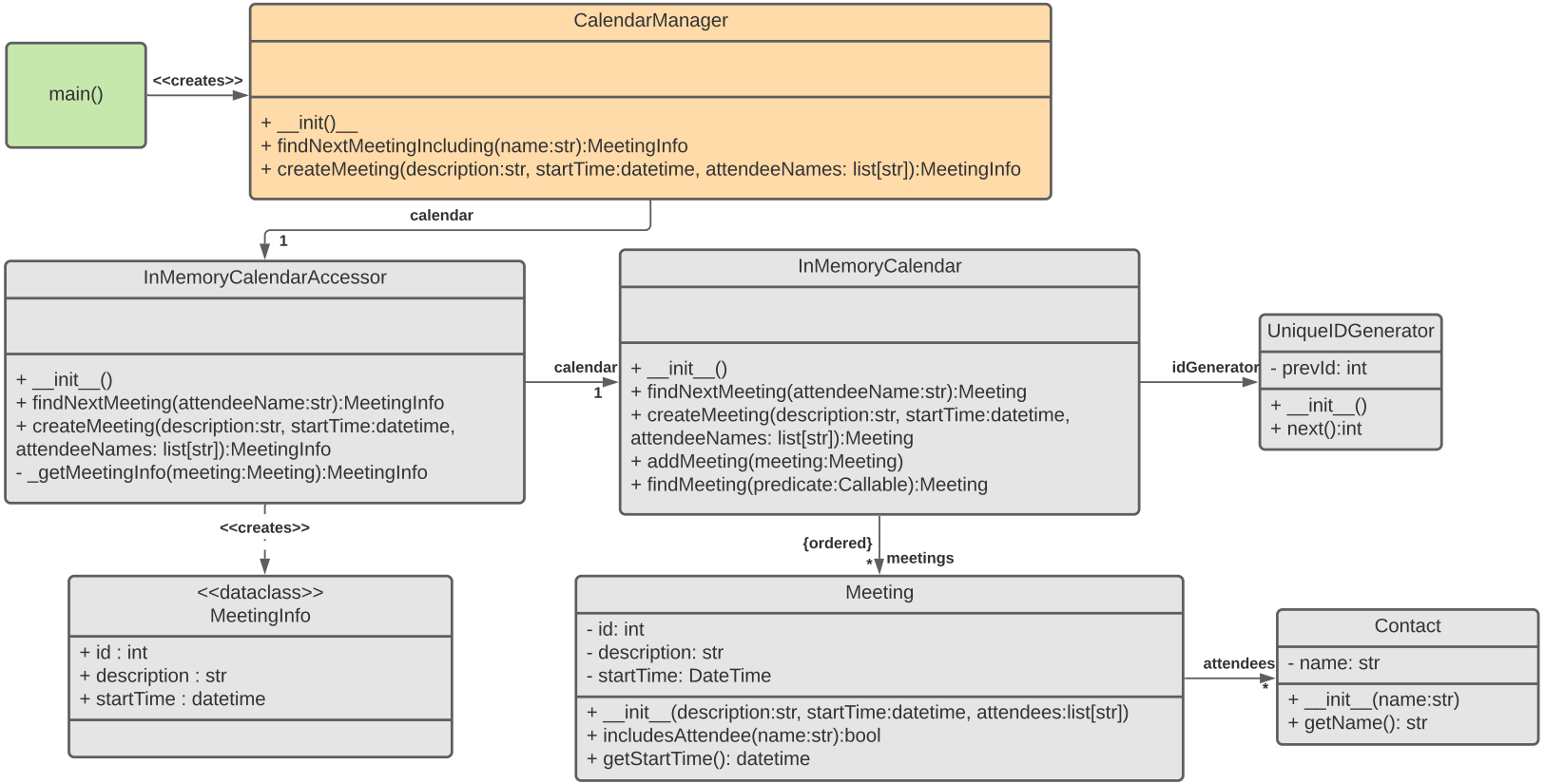
In VS Code terminal, use py instead of the full path to python.exe

Task 2. Test the Calendar Application

- We will revisit the implementation of the Calendar application from Practical 1 to test the classes implemented in Practical 1

- You can work with your code or download the provided code for Practical 2 from the course site.

Practical 1 Implementation Design



Test findNextMeeting() #1

- Rewrite the test performed in the main() method as pytest test.
  - Name the test function *test_findNextMeeting*
  - Put all test functions in a separate source file, not intermingled with the rest of the code.

- Run pytest to verify that the test passes.
- Then delete the main() method from the program and verify again that the test still works.

```
def test_findNextMeeting():
    manager = CalendarManager()
    manager.createMeeting("First Meeting", datetime(2021,5,17,9,10), ["Wolfgang"])
    manager.createMeeting("Second Meeting", datetime(2021,5,17,11,10),
                          ["Wolfgang","Mei","Matt"])
    manager.createMeeting("Third Meeting", datetime(2021,5,17,14,10), ["Matt"])
    meeting = manager.findNextMeeting("Matt")
    assert meeting.description == "Second Meeting"
```

Test findNextMeeting() #2

- Write a test that verifies that findNextMeeting() returns None if there is no meeting that includes the given attendee's name.
- Use the same three meetings as in the previous test.
- Name the test *test_findNextMeeting_NoneFound*

```
def test_findNextMeeting_NoneFound():
    manager = CalendarManager()
    manager.createMeeting("First Meeting", datetime(2021,5,17,9,10),
                          ["Wolfgang"])
    meeting = manager.findNextMeeting("Matt")
    assert meeting is None
```

Test Fixture #1

- Create a test fixture that creates the Manager object.
- Rewrite the previous two tests to use the fixture.
- Verify that the tests pass.

```python
@pytest.fixture
def manager():
    return CalendarManager()

def test_findNextMeeting_NoneFound(manager):
    manager.createMeeting("First Meeting", datetime(2021,5,17,9,10),
                          ["Wolfgang"])
    meeting = manager.findNextMeeting("Matt")
    assert meeting is None


def test_findNextMeeting(manager):
    ...
    assert meeting.description == "Second Meeting"
```

Test Fixture #2

- Create a test fixture that creates the same three meetings as in the previous tests.
- Rewrite the previous two tests to use the fixture (in addition to the Manager fixture created earlier).
- Verify that the tests pass.

```python
@pytest.fixture
def scenario1_meetings_manager(manager):
    manager.createMeeting("First Meeting", datetime(2021,5,17,9,10),
                          ["Wolfgang"])
    manager.createMeeting("Second Meeting", datetime(2021,5,17,11,10),
                          ["Wolfgang","Mei","Matt"])
    manager.createMeeting("Third Meeting", datetime(2021,5,17,14,10),
                          ["Matt"])
    return manager
```

```
def test_findNextMeeting_NoneFound(scenario1_meetings_manager):
    scenario1_meetings_manager.createMeeting("First Meeting",
                          datetime(2021,5,17,9,10), ["Wolfgang"])
    meeting = scenario1_meetings_manager.findNextMeeting("Matt")
    assert meeting is None


def test_findNextMeeting(scenario1_meetings_manager):
    ...
    assert meeting.description == "Second Meeting"
```

Test Meeting Requires Attendees

- Write a test to verify that that creating a meeting without attendees raises a *ValueError* exception.

- Name the test *test_Meeting_Zero_Attendees*

- This test may fail initially.
- Debug the program and verify that the test passes.

```
def test_meeting_must_have_attendees(manager):
    with pytest.raises(ValueError):
        manager.createMeeting("Meeting nobody attends",
                              datetime(2021,5,17,1,0), [])
```

```
class Meeting:
    def __init__(self, description, startTime, attendeeNames):
        if not attendeeNames:
            raise ValueError("Meeting must have one or more " \
                             "attendees")
        self.id = self.idGenerator.next()
        self.description = description
        self.attendees = [self._createAttendee(name)
                          for name in attendeeNames]
        self.startTime = startTime
```

Test findNextMeeting() #3

- Create another test fixture that creates the following meetings (in this order):

Description	Start Time	Attendees
Morning meeting	9am on 24 May	Alice, Bob
Afternoon meeting	3pm on 24 May	Bob, Charlie
Lunch meeting	12pm on 24 May	Alice, Bob, Charlie

```python
@pytest.fixture
def scenario2_meetings_manager(manager):
    manager.createMeeting("Morning Meeting", datetime(2021,5,24,9,10), ["Alice"])
    manager.createMeeting("Afternoon Meeting", datetime(2021,5,24,15,0),
                          ["Bob","Charlie"])
    manager.createMeeting("Lunch Meeting", datetime(2021,5,24,12,0),
                          ["Alice","Bob","Charlie"])
    return manager

def test_find_next_meeting_reordered(scenario2_meetings_manager):
    meeting = scenario2_meetings_manager.findNextMeeting(”Charlie")
    assert meeting.description == "Lunch Meeting"
```

Test findNextMeeting() #3 (cont.)

- Test that findNextMeeting() returns the *Lunch Meeting* as the next meeting that includes *Charlie*

- If this test fails, debug the program and verify that the bug has been resolved.

```
class InMemoryCalendar:

    def addMeeting(self, meeting):
        self.meetings.append(meeting)
        # We must keep the meetings sorted because findNextMeeting()
        # relies on it
        self.meetings.sort(key=lambda meeting: meeting.getStartTime())
```

Task 3. Pull Out Dependencies

- We wish to prepare our code for the introduction of an alternative calendar backend backed by a database.
- Would the current implementation support this easily?
  - Why/Why not?

Issue

The CalendarManager creates its collaborator, the Accessor:

```
class CalendarManager:

    def __init__(self):
        self.calendar = InMemoryCalendarAccessor()
```

This is undesirable. How to resolve this?

Dependency Injection

- Apply the dependency injection principle so that accessor is supplied as an argument to the constructor of the CalendarManager.
- Refactor the tests to construct the adapter and pass it to the Manager.
- Verify that all tests pass.

```
class CalendarManager:

def __init__(self, calendarAccessor):
    self.calendar = calendarAccessor

@pytest.fixture
def manager():
    accessor = InMemoryCalendarAccessor()
    return CalendarManager(accessor)
```

Task 4. Test Stub

- Suppose we want to test the implementation of the CalendarManager class, but we have not yet implemented the Accessor class.

- Use a Mock object as a double for the Accessor object.
- Test that findNextMeeting() in the Accessor object is indeed called by findNextMeeting() in the Manager object.

```
def test_mgr_findNextMeeting_calls_accessor():
    accessor = Mock()
    manager = CalendarManager(accessor)
    manager.findNextMeeting("Alice")
    assert accessor.findNextMeeting.called_once_with("Alice")
```

```
def test_mgr_findNextMeeting_result():
    expected = MeetingInfo("Nonexistant meeting",
                           datetime(2021,5,24,7,0),["Alice"])
    accessor = Mock()
    accessor.findNextMeeting.return_value = expected
    manager = CalendarManager(accessor)
    meeting = manager.findNextMeeting("Alice")
    assert meeting == expected
```

You Should Know

- Write Unit Tests for your code
- Create stub/mock objects
- Apply Test-Driven Development practices

Activities this Week

- Complete Quiz 7
- Start working on Assignment 2

University of
South Australia