Model View Presenter for testable maintainable PyQt GUIs

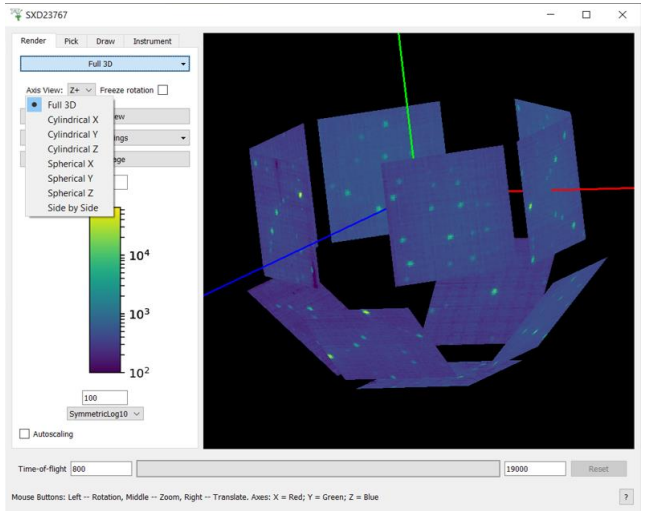
Sam Tygier

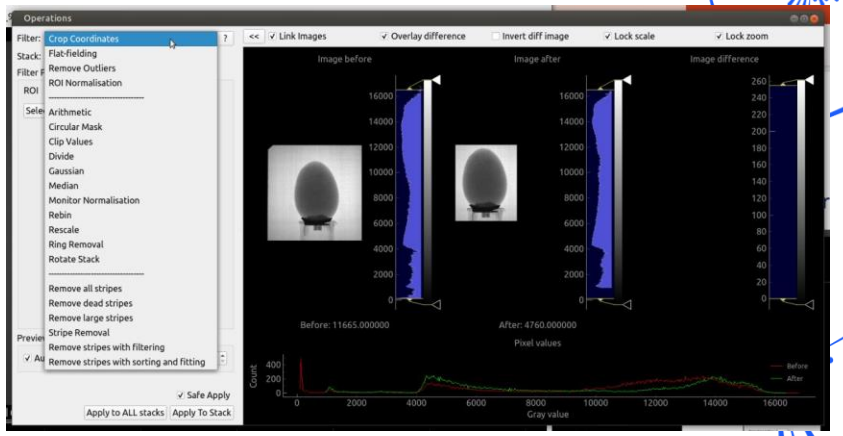
4 Feb 2025

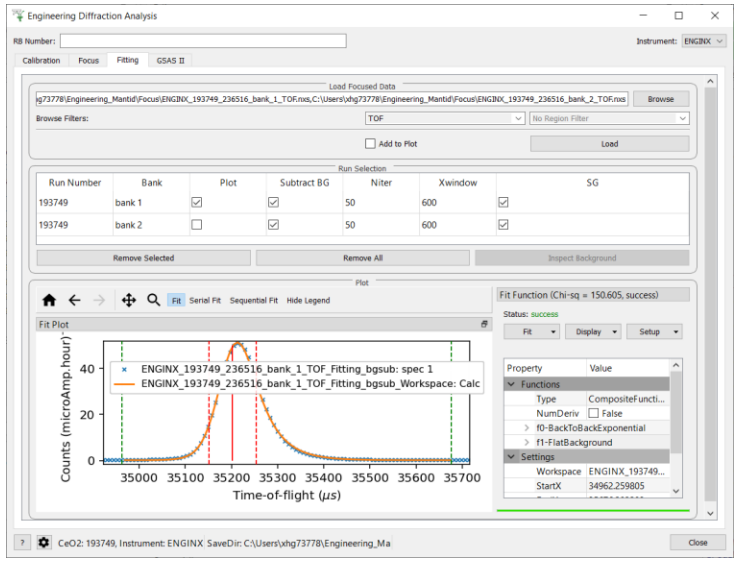
Intro

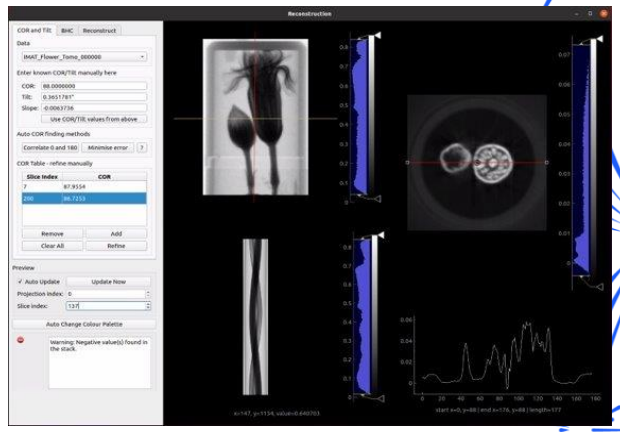
- Complexity
- Model View Presenter
- Maintainability
- Testing
- Practicalities

Based on experience with Qt
mostly in python (some C++)
PyQt or PySide
Focus on desktop applications
Some general lessons



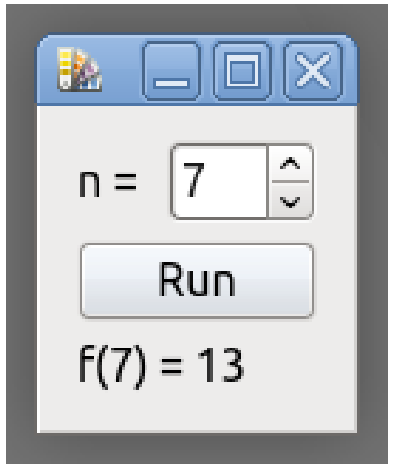






Simple PyQt

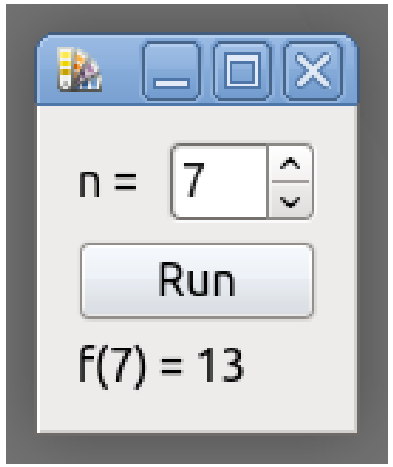
- PyQt examples
  - Create an application
  - Create a window
  - Add some widgets
  - Define a function
  - Connect a signal
  - Run




```
app = QApplication([])
window = QWidget()
layout = QFormLayout(window)
n_input = QSpinBox()
n_input.setValue(1)
run_button = QPushButton("Run")
f_output = QLabel("")
layout.insertRow(0, QLabel("n = "), n_input)
layout.insertRow(1, run_button)
layout.insertRow(2, f_output)

def run_fib():
    n = n_input.value()
    fib = [0,1]
    while len(fib) <= n:
        fib.append(fib[-1] + fib[-2])
    f_output.setText(f"f({n}) = {fib[n]}")

run_button.clicked.connect(run_fib)

window.show()
app.exec()
```

Simple PyQt - Class

- PyQt examples
  - Create an application
  - Create a window
  - Add some widgets
  - Define a function
  - Connect a signal
  - Run
- Or with a class
  - Constructor
    - Create UI
    - Connect signals
  - Methods
    - Do work




```
class FibWindow(QWidget):
    def __init__(self):
        super().__init__()
        layout = QFormLayout(self)
        self.n_input = QSpinBox()
        self.n_input.setValue(1)
        run_button = QPushButton("Run")
        self.f_output = QLabel("")
        layout.insertRow(0, QLabel("n = "),
self.n_input)
        layout.insertRow(1, run_button)
        layout.insertRow(2, self.f_output)

        run_button.clicked.connect(self.run_fib)

    def run_fib(self):
        n = self.n_input.value()
        fib = [0,1]
        while len(fib) <= n:
            fib.append(fib[-1] + fib[-2])
        self.f_output.setText(f"f({n}) = {fib[n]}")

if __name__ == '__main__':
    app = QApplication([])
    window = FibWindow()
    window.show()
    app.exec()
```

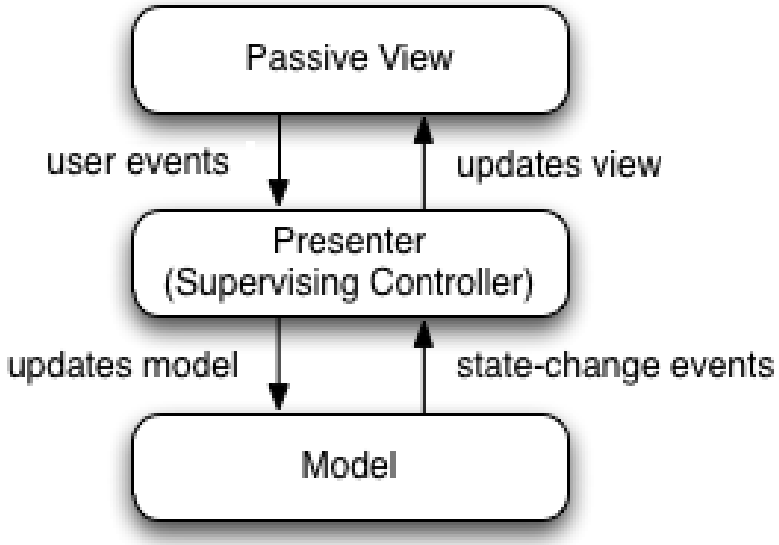
Problems

- How to test
  - What can I unit test?
  - Can I run tests without starting the GUI?
- What happens as this grows
  - One giant class?
  - One giant file?
- Making changes
  - Change behavior?
  - Change GUI design?
  - Change algorithm?
  - Change GUI toolkit?

```
class FibWindow(QWidget):
    def __init__(self):
        super().__init__()
        layout = QFormLayout(self)
        self.n_input = QSpinBox()
        self.n_input.setValue(1)
        run_button = QPushButton("Run")
        self.f_output = QLabel("")
        layout.insertRow(0, QLabel("n = "), self.n_input)
        layout.insertRow(1, run_button)
        layout.insertRow(2, self.f_output)

        run_button.clicked.connect(self.run_fib)

    def run_fib(self):
        n = self.n_input.value()
        fib = [0,1]
        while len(fib) <= n:
            fib.append(fib[-1] + fib[-2])
        self.f_output.setText(f"f({n}) = {fib[n]}")

if __name__ == '__main__':
    app = QApplication([])
    window = FibWindow()
    window.show()
    app.exec()
```

Model View Presenter (MVP)

- Design pattern
- Separation of concerns
- **View**
  - Simple passive code
  - Layout, design
- **Presenter**
  - Business logic
  - Received events from GUI
  - Interacts with Model
  - Updates GUI
- **Model**
  - Domain logic
  - Where the work is done
  - Algorithms / Database



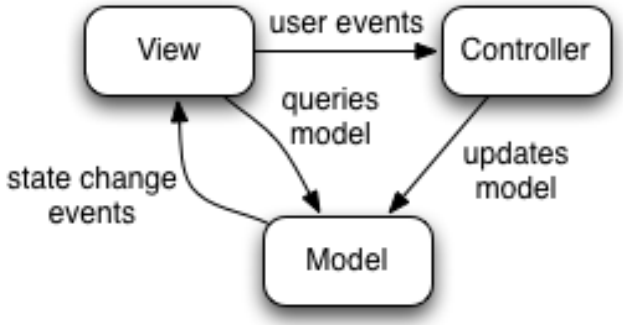
- No direct communication between model and view
- Model has no knowledge of GUI
- View has no logic
- Presenter links everything together

Bad name:
Confused with Minimum Viable Product
Should be Model-Presenter-View

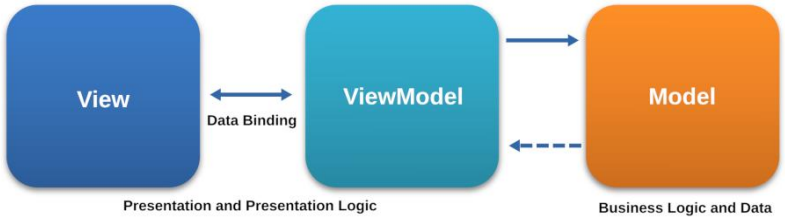
- Google - http://www.gwtproject.org/articles/testing_methodologies_using_gwt.html

Similar things

- There are selection of related patterns
  - Model–view–controller
  - Model–view–viewmodel
  - Model-view-template
- Names not always used consistently
- Some practical tech differences
  - Web frameworks
  - IDEs and programming environments
- General principals
  - Separations of concerns
  - Well defined communications
  - Reusability



Model View Controller



Model View Viewmodel

GUI Architectures – Martin Fowler
https://www.martinfowler.com/eaaDev/uiArchs.html

- Google - http://www.gwtproject.org/articles/testing_methodologies_using_gwt.html
- Wikipedia Uncopy - https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

MVP for Python Qt

Model	Presenter	View
• Code that does work • Domain objects • Unit Testable • No knowledge of GUI • No Qt types	• Handles all comms between Model and View • "Business logic" • Testable with mocking	• Display and layout • GUI widgets • No/Low logic • System/screenshot tests • Different views for different OS / Toolkits

Separation of concerns

Layout

How does it look

Domain logic

The science model

```
class FibWindow(QWidget):
    def __init__(self):
        super().__init__()
        layout = QFormLayout(self)
        self.n_input = QSpinBox()
        self.n_input.setValue(1)
        run_button = QPushButton("Run")
        self.f_output = QLabel("")
        layout.insertRow(0, QLabel("n = "), self.n_input)
        layout.insertRow(1, run_button)
        layout.insertRow(2, self.f_output)

        run_button.clicked.connect(self.run_fib)

    def run_fib(self):
        n = self.n_input.value()
        fib = [0,1]
        while len(fib) <= n:
            fib.append(fib[-1] + fib[-2])
        self.f_output.setText(f"f({n}) = {fib[n]}")

if __name__ == '__main__':
    app = QApplication([])
    window = FibWindow()
    window.show()
    app.exec()
```

Business logic

Behavior of application
When user clicks "Run", take the value from input form, calculate value, display result in output.

MVP - Model

- Pure domain code
- No GUI related code
- Easy to write unit tests
- Easy to change algorithm

```python
class FibModel:
    def run_fib(self, n: int) -> int:
        fib = [0,1]
        while len(fib) <= n:
            fib.append(fib[-1] + fib[-2])
        return fib[n]
```

```python
class FibUnitTest(unittest.TestCase):
    @parameterized.expand([(1, 1), (2, 1), (3, 2), (10, 55)])
    def test_fib_n(self, n, expected_f):
        model = FibModel()
        self.assertEqual(model.run_fib(n), expected_f)
```

MVP - view

- Pure GUI code
- No/Low logic
- Hard to test
  - But not many places for bugs to hide
- Does not need to know much about the presenter

- C++
  - Handle QString <-> std::string

```
class FibWindowView(QWidget):
    presenter: BasePresenter
    def __init__(self):
        super().__init__()
        layout = QFormLayout(self)
        self.n_input = QSpinBox()
        self.n_input.setValue(1)
        run_button = QPushButton("Run")
        self.f_output = QLabel("")
        layout.insertRow(0, QLabel("n = "), self.n_input)
        layout.insertRow(1, run_button)
        layout.insertRow(2, self.f_output)

        run_button.clicked.connect(self._handle_run_button)

    def set_presenter(self, presenter: BasePresenter):
        self.presenter = presenter

    def _handle_run_button(self):
        self.presenter.notify(Notification.RUN_CALC)

    def get_n(self) -> int:
        return self.n_input.value()

    def set_output(self, result: str):
        self.f_output.setText(result)
```

MVP - presenter

- Glues everything together
- Knows about the view and model
- Coordinates
  - Receiving user inputs
  - Getting information from view
  - Doing work in the model
  - Updating the view
- Does not need to know about GUI toolkit
  - FibWindowView could be an interface, with subclasses for Qt, GTK, Cocoa

```
class FibWindowPresenter(BasePresenter):
    def __init__(self, view: FibWindowView, model: FibModel):
        self.view = view
        self.model = model
        self.view.set_presenter(self)


    def notify(self, event: Notification):
        match event:
            case Notification.RUN_CALC:
                self.run_calculation()

    def run_calculation(self):
        n = self.view.get_n()
        f = self.model.run_fib(n)
        result = f"f({n}) = {f}"
        self.view.set_output(result)
```

Practicalities

- View and presenter hold references/pointers to each other
  - In a strongly typed language this may not be possible directly
  - Have a BasePresenter with very narrow interface: notification method
  - In python not necessary, can call presenter methods from view
  - Weak references

- Where to hold state
  - In example we retrieve n from view when needed
  - Could update n in model when it changes
  - Risk of getting state out of sync

- Where to instantiate classes
  - Create M, V and P outside and link them
    - Easier to pass mock in during testing
  - Let V create P, and P create M
    - Simpler from outside
- Logic in view
  - Can save a lot of back and forth calls

- Qt in presenter and model
  - Qt has more than just GUI tools
  - e.g. QFileSystemWatcher, QThread

No notify version

With notify

```
class FibWindowView(QWidget):
    presenter: BasePresenter
    def __init__(self):
        super().__init__()
        layout = QFormLayout(self)
        self.n_input = QSpinBox()
        self.n_input.setValue(1)
        run_button = QPushButton("Run")
        self.f_output = QLabel("")
        layout.insertRow(0, QLabel("n = "), self.n_input)
        layout.insertRow(1, run_button)
        layout.insertRow(2, self.f_output)

        run_button.clicked.connect(self._handle_run_button)

    def set_presenter(self, presenter: BasePresenter):
        self.presenter = presenter

    def _handle_run_button(self):
        self.presenter.notify(Notification.RUN_CALC)
```

Without notify

```
class FibWindowView(QWidget):
    presenter: FibWindowPresenter
    def __init__(self):
        super().__init__()
        layout = QFormLayout(self)
        self.n_input = QSpinBox()
        self.n_input.setValue(1)
        run_button = QPushButton("Run")
        self.f_output = QLabel("")
        layout.insertRow(0, QLabel("n = "), self.n_input)
        layout.insertRow(1, run_button)
        layout.insertRow(2, self.f_output)

        run_button.clicked.connect(self._handle_run_button)

    def set_presenter(self, presenter: FibWindowPresenter):
        self.presenter = presenter

    def _handle_run_button(self):
        self.presenter.run_calculation()
```

- Easier to pass arguments
- Fewer function calls
- Less boilerplate
- Better static analysis

No notify version 2

Pass value

```
class FibWindowView(QWidget):
    presenter: FibWindowPresenter
    def __init__(self):
        super().__init__()
        layout = QFormLayout(self)
        self.n_input = QSpinBox()
        self.n_input.setValue(1)
        run_button = QPushButton("Run")
        self.f_output = QLabel("")
        layout.insertRow(0, QLabel("n = "), self.n_input)
        layout.insertRow(1, run_button)
        layout.insertRow(2, self.f_output)

        run_button.clicked.connect(self._handle_run_button)

    def set_presenter(self, presenter: FibWindowPresenter):
        self.presenter = presenter

    def _handle_run_button(self):
        n = self.get_n()
        self.presenter.run_calculation(n)
```

Without handle method

```
class FibWindowView(QWidget):
    presenter: FibWindowPresenter
    def __init__(self):
        super().__init__()
        layout = QFormLayout(self)
        self.n_input = QSpinBox()
        self.n_input.setValue(1)
        run_button = QPushButton("Run")
        self.f_output = QLabel("")
        layout.insertRow(0, QLabel("n = "), self.n_input)
        layout.insertRow(1, run_button)
        layout.insertRow(2, self.f_output)

        run_button.clicked.connect(self.presenter.run_calculation)

    def set_presenter(self, presenter: FibWindowPresenter):
        self.presenter = presenter
```

- Less back and forth
- Fewer pass-through methods

Testing the GUI

- System tests
- Launch the application
- Use QTest to interact with UI

```
class FibSystemTest(unittest.TestCase):
    def setUp(self) -> None:
        self.view = FibWindowView()
        self.model = FibModel()
        self.presenter = FibWindowPresenter(self.view, self.model)

        self.view.show()
        QTest.qWait(10)

    def tearDown(self):
        QTest.qWait(10)
        self.view.close()

    def test_calc_fib(self):
        QTest.keySequence(self.view.n_input, QKeySequence.SelectAll)
        QTest.keyClicks(self.view.n_input, "10")
        QTest.mouseClick(self.view.run_button, Qt.LeftButton)
        self.assertEqual("f(10) = 55", self.view.f_output.text())
```

- Screen shot tests
- Save image of GUI
- Need other tools for comparison

```
class FibScreenShotTest(unittest.TestCase):
    def setUp(self) -> None:
        ...

    def tearDown(self):
        ...

    def _take_screen_shot(self, widget:QWidget, filename: str):
        QTest.qWaitForWindowExposed(widget)
        QApplication.sendPostedEvents()
        QApplication.processEvents()
        window_image = widget.grab()
        window_image.save(filename, "PNG")

    def test_calc_fib_new_window(self):
        self._take_screen_shot(self.view, "fib.png")

    def test_calc_fib_changed(self):
        QTest.keySequence(self.view.n_input, QKeySequence.SelectAll)
        QTest.keyClicks(self.view.n_input, "10")
        QTest.mouseClick(self.view.run_button, Qt.LeftButton)
        self._take_screen_shot(self.view, "fib_n10.png")
```

Same as system test

More info

- Code examples
  - https://github.com/samtygier-stfc/mvp-talk-examples
- Mantid training
  - https://developer.mantidproject.org/MVPDesign.html
  - https://developer.mantidproject.org/MVPTutorial/index.html
- Martin Fowler
  - https://www.martinfowler.com/eaaDev/uiArchs.html