HOW CAN TEST-DRIVEN DEVELOPMENT FIT WITH RESEARCH SOFTWARE?

WHAT IS TEST-DRIVEN DEVELOPMENT?

SKETCH

framer.py:

```
def get_n_frames(n, hop):

  """Return the number of
     non-overlapping frames
     of length hop we can
     get from n samples."""

  return 0
```

WRITE TESTS

test_framer.py:

```
import framer as fr;

def test_get_n_frames():
  assert fr.get_n_frames(4,2) == 2
  assert fr.get_n_frames(5,2) == 3
  assert fr.get_n_frames(6,1) == 6
  assert fr.get_n_frames(0,5) == 0
```

RUN TESTS

console:

```
$ nosetests
==============================
FAIL:
test_framer.test_get_n_frames
------------------------------
  assert fr.get_n_frames(4,2) == 2
AssertionError
```

FILL IN CODE

framer.py:

```
def get_n_frames(n, hop):

  """Return the number of
     non-overlapping frames
     of length hop we can
     get from n samples."""

  f = int(np.ceil(n/float(hop)))
  return f
```

WHAT DO WE WANT THIS BIT OF CODE TO DO? HOW WILL WE USE IT?

THESE ARE HERE SO WE CAN TELL WHEN WE'VE FINISHED WRITING THE CODE

THE TESTS SHOULD RUN... BUT FAIL

GO ROUND AND ROUND WRITING CODE FOR THIS FUNCTION UNTIL IT PASSES THE TESTS

HOW DO YOU DO THAT WITH RESEARCH CODE?

LOW-LEVEL STUFF

When the task is clear enough that you can work out the answers "by hand" for simple inputs:

- Use simple synthetic test data, not real-world data
- Include trivial or null inputs
- Test any utility code like framing, file I/O
- What's the simplest input that makes it fail?

Try to arrange work into units small enough to be reasoned about this way

EXPERIMENTAL STUFF

Not easy to work out what results are expected?

- You're testing the implementation—not the quality of the method it uses (that's for the paper)
- Seek "minimum valid behaviour" for method
- Existing methods, no matter how bad, should pass your tests if implemented correctly

Tests provide the basic code safety net, enabling wild experimental changes without screwing up

WATCH

Video from ISMIR 2012 tutorial
http://bit.ly/UrPFGF

Code written during that tutorial
http://bit.ly/SLfMuD

READ

Real-world C++ audio analysis example
http://bit.ly/Z4XpWE

More handouts and guides
http://bit.ly/SLfUuc

EPSRC
Pioneering research and skills

soundsoftware.ac.uk