

Open Source MANO

Writing Good Tests Mark Beierl (Canonical)

What is Legacy Code?

- Spaghetti Code
- Poorly Structured
- Not documented, or misleading comments
- “Someone else’s code”
- Code *without tests*
 - With tests we can change quickly, and verify
 - Without, we don’t know if it’s better or worse

From Working Effectively With Legacy Code
Michael C. Feathers

What is a Unit Test?

- Different opinions:
 - Method level?
 - If clause level?
 - Success path / failure path?
 - Automated, or manual set up?
 - Special environment to run?
- If the meaning does not match intent, do we know what to do?

A new term: Micro Test

What is a Micro Test?

- Short, few lines of code
- Always automated
- Purpose built test application
- Test a single branch of logic
- Test code written to same standard as regular code
- Test code is in git too
- Serves as gateway to commit
- Very quick
 - milliseconds per test
- Precise feedback on errors
- Part of a collection
- Easy to invoke
- Grey box
 - Can manipulate contents if needed
- Avoids use of collaborators through the use of mock or stub objects
- Involves creation of very few objects
- Does not require any external software

From *They're Called Microtests*
<http://anarchycreek.com/2009/05/20/theyre-called-microtests/>

- What do I test?
 - Expected behaviour
 - Logic paths
 - External API
 - Exceptions
 - **Impossible** conditions
- What don't I test?
 - Things that are too simple to break?
 - Getters / Setters
- When have I tested enough?
 - When *fear* turns to *boredom*...

Tests as Documentation

- A good test demonstrates:
 - Functionality
 - Expected inputs / outputs
 - Exception handling
 - Interactions with other objects
- Tests can serve as a document about how to use the API
 - Example of how to use the function under test
 - What types of exceptions can happen

Idempotent and Independent

- Tests must:
 - Be self-contained
 - Be repeatable
 - Have everything needed to cover all logic paths
- Tests must not:
 - Cause changes in the environment
 - Leave anything behind
 - Depend on prior test execution
 - Have any side effects
 - Launch a rocket



But my Function Launches a Rocket!

How can we test a rocket without sending it into orbit?

MOCK IT

```
@mock.patch.object(GnocchiBackend, '_build_neutron_client')
@mock.patch.object(GnocchiBackend, '_build_gnocchi_client')
def test_collect_gnocchi_non_rate_instance(self, build_gnocchi_client, _):
```


What is a Mock?

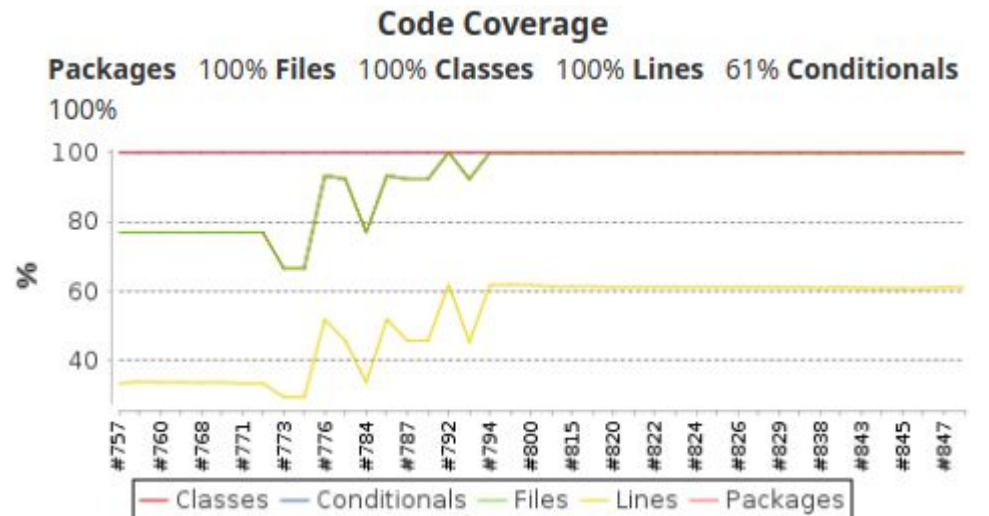
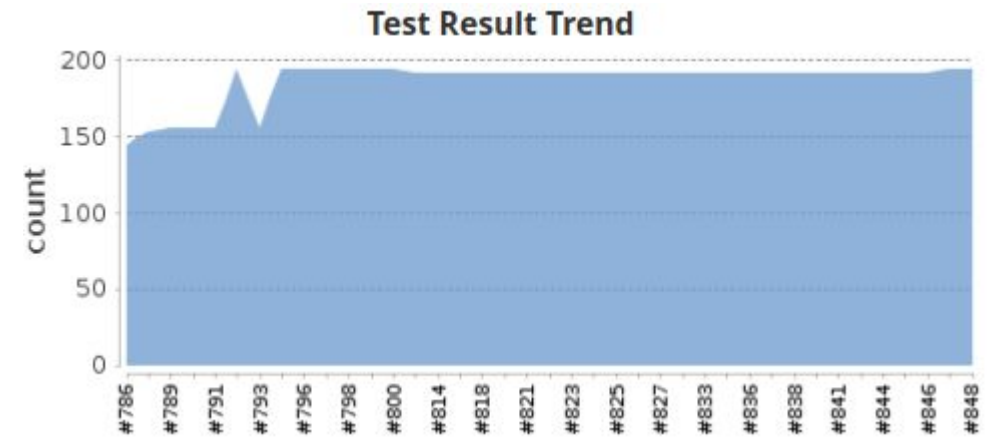
- Simulated objects that mimic the behavior of real objects in controlled ways
- Use a mock if the object
 - Has non-deterministic results
 - (e.g. the current time or the current temperature)
 - has states that are difficult to create or reproduce
 - (e.g. a network error)
 - is slow
 - (e.g. a complete database, which would have to be initialized before the test)

What Can a Mock Do?

- It does only what it is told to do, nothing more
- Can return any value
 - `mock.side_effect = "123"`
- Can throw exceptions
 - Even “impossible ones”
 - `mock.side_effect = DatabaseIndexCorruptedException()`
- No logic path or exception handler should go without testing!

Proof?

- Already part of our pipeline
- Both
 - Pre-merge commits
 - Post-merge commits



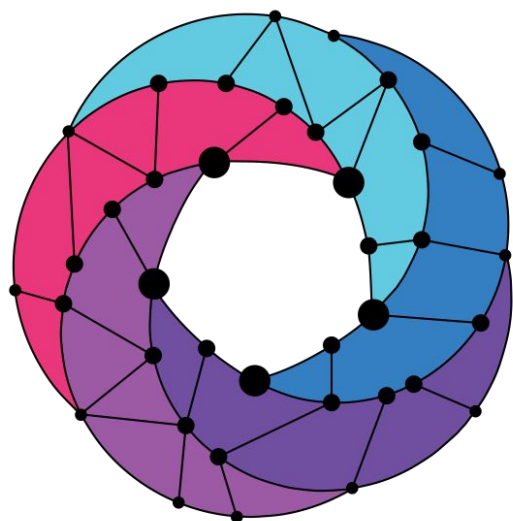
```
196 |         # Block until other workers have finished model creation
197 | 1   while self.creating_model.locked():
198 | 0   await asyncio.sleep(0.1)
199 |
200 |         # If the model exists, return it from the controller
201 | 1   if model_name in self.models:
202 | 0   return
203 |
204 |         # Create the model
205 | 1   async with self.creating_model:
206 | 1       self.log.debug("Creating model {}".format(model_name))
207 | 1       model = await controller.add_model(
```

Protect the Code

- Putting it all together:
 - Lots of very fast micro tests
 - Covering a predefined percentage of the code base
 - ... or Jenkins could the job
- A perfect companion to Gerrit
 - Pre-review gate (the stage 2 job)
 - Reviews can be rejected if
 - A test is broken
 - The percentage of code coverage drops
- Prevents “Legacy Code”

What Have We Learned?

- Code without tests is tomorrow's legacy code
- Microtests = "Good Unit Tests"
 - Fast, repeatable, Idempotent, Independent
- Mocks replace slow, dangerous or difficult collaborators
- There is no code that is too complex to test
- Jenkins knows how to read unit test and code coverage results
- Gerrit can prevent patches that violate the norms set by the project



Open Source MANO

Find us at:

osm.etsi.org
osm.etsi.org/wikipub