```
import unittest
import os
from unittest.mock import patch, mock_open
import tempfile
import json
from swarms.artifacts.main_artifact import Artifact


class TestArtifactSaveAs(unittest.TestCase):
    def setUp(self):
        """Set up test fixtures before each test method."""
        self.temp_dir = tempfile.mkdtemp()
        self.test_file_path = os.path.join(
            self.temp_dir, "test_file.txt"
        )
        self.test_content = (
            "This is test content\nWith multiple lines"
        )

        # Create artifact with all required fields
        self.artifact = Artifact(
            file_path=self.test_file_path,
            file_type=".txt",
            contents=self.test_content,  # Provide initial content
            edit_count=0,
        )
```

```
    self.artifact.create(self.test_content)


def tearDown(self):
    """Clean up test fixtures after each test method."""
    try:
        if os.path.exists(self.test_file_path):
            os.remove(self.test_file_path)
        # Clean up any potential output files
        base_path = os.path.splitext(self.test_file_path)[0]
        for ext in [".md", ".txt", ".py", ".pdf"]:
            output_file = base_path + ext
            if os.path.exists(output_file):
                os.remove(output_file)
        os.rmdir(self.temp_dir)
    except Exception as e:
        print(f"Cleanup error: {e}")


def test_save_as_txt(self):
    """Test saving artifact as .txt file"""
    output_path = (
        os.path.splitext(self.test_file_path)[0] + ".txt"
    )
    self.artifact.save_as(".txt")
    self.assertTrue(os.path.exists(output_path))
    with open(output_path, "r", encoding="utf-8") as f:
        content = f.read()
```

```
    self.assertEqual(content, self.test_content)

def test_save_as_markdown(self):
    """Test saving artifact as .md file"""
    output_path = os.path.splitext(self.test_file_path)[0] + ".md"
    self.artifact.save_as(".md")
    self.assertTrue(os.path.exists(output_path))
    with open(output_path, "r", encoding="utf-8") as f:
        content = f.read()
    self.assertIn(self.test_content, content)
    self.assertIn("# test_file.txt", content)

def test_save_as_python(self):
    """Test saving artifact as .py file"""
    output_path = os.path.splitext(self.test_file_path)[0] + ".py"
    self.artifact.save_as(".py")
    self.assertTrue(os.path.exists(output_path))
    with open(output_path, "r", encoding="utf-8") as f:
        content = f.read()
    self.assertIn(self.test_content, content)
    self.assertIn("'''", content)
    self.assertIn("Generated Python file", content)

@patch("builtins.open", new_callable=mock_open)
def test_file_writing_called(self, mock_file):
    """Test that file writing is actually called"""
```

```
    self.artifact.save_as(".txt")
    mock_file.assert_called()
    mock_file().write.assert_called_with(self.test_content)

def test_invalid_format(self):
    """Test saving artifact with invalid format"""
    with self.assertRaises(ValueError):
        self.artifact.save_as(".invalid")

def test_export_import_json(self):
    """Test exporting and importing JSON format"""
    json_path = os.path.join(self.temp_dir, "test.json")

    # Export to JSON
    self.artifact.export_to_json(json_path)
    self.assertTrue(os.path.exists(json_path))

    # Import from JSON and convert timestamp back to string
    with open(json_path, "r") as f:
        data = json.loads(f.read())
        # Ensure timestamps are strings
        for version in data.get("versions", []):
            if isinstance(version.get("timestamp"), str):
                version["timestamp"] = version["timestamp"]

    # Import the modified data
```

```
        imported_artifact = Artifact(**data)
        self.assertEqual(
            imported_artifact.contents, self.test_content
        )

        # Cleanup
        os.remove(json_path)


if __name__ == "__main__":
    unittest.main()
```