UNIVERSITY
OF COLOGNE

DLR

PROPERTY-BASED TESTING OF DATA ANALYSIS SCRIPTS

A Focus on Hypothesis for Python

By Jean-Sebastian de Wet, Jan-Philipp Kiel and Pascal Mager of Group Criterion

17.01.2024

1

INTRODUCTION

The Evolving Data Analysis Landscape

Exponential Data Growth

Rapid increase in complexity and volume of datasets

Need for Robust Testing

Requires robust test methods to ensure accuracy and reliability

Traditional Testing Limits

Shortcomings of case-based testing for dynamic datasets

Aerospace Research

Challenges due to the immense scope and variety of data

2

BACKGROUND AND METHODOLOGY

Background:
Data Analysis at the German Aerospace Center (DLR)

Reliability Focus

Crucial for DLR's research accuracy and success

Python and Libraries

Usage of Python, Pandas and Matplotlib for complex tasks

Diverse Data Range

Wide array of data from satellite imagery to flight dynamics

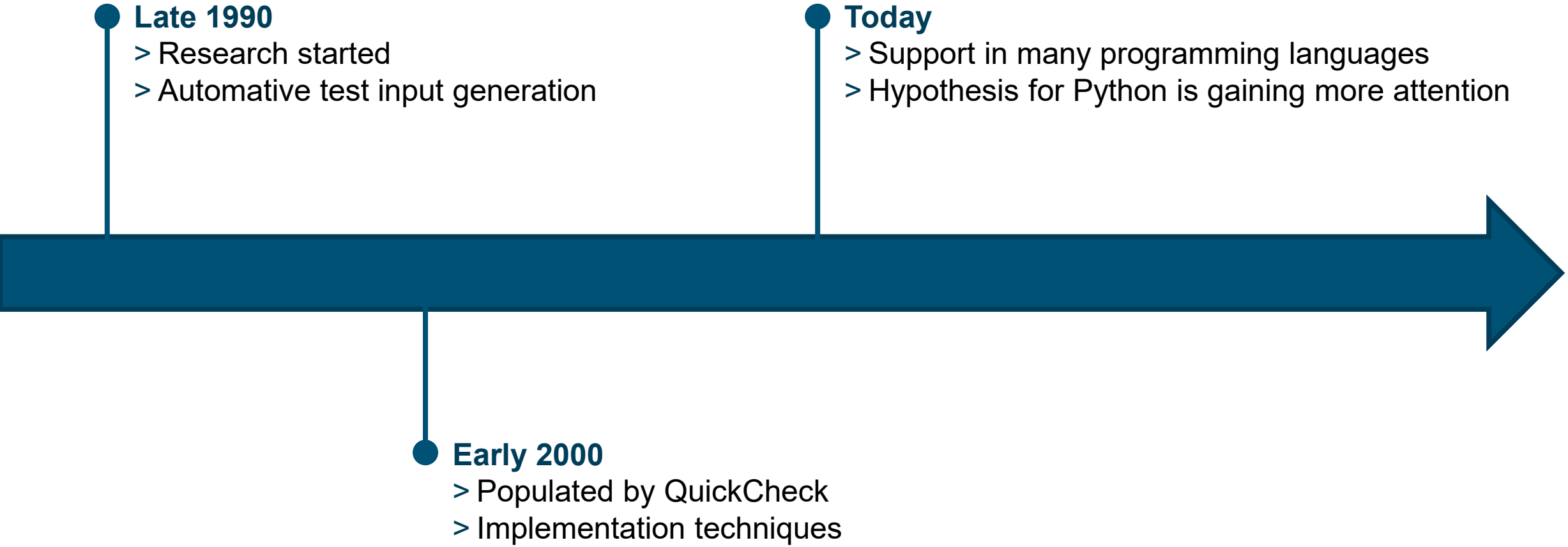
Methodology:
Literature Study and Prototyping

Literature Study	Prototyping
> Research concept: Exploring property-based testing principles > Focus on primary literature: Selection of highly cited sources, such as from the ACM database > Inclusion of secondary literature: Expansion of research to include additional relevant studies	> Development of a tutorial: Simple guide for Hypothesis for Python > Case study on astronaut data: Property-based testing in a practical scenario

3

LITERATURE STUDY OF PROPERTY-BASED TESTING

History of Property-Based Testing



Key Concepts of Property-Based Testing

Properties

General behavioral conditions for wide input range

Generative Testing

Automated generation of inputs for test execution

Randomized Input Generation

Create random comprehensive test inputs

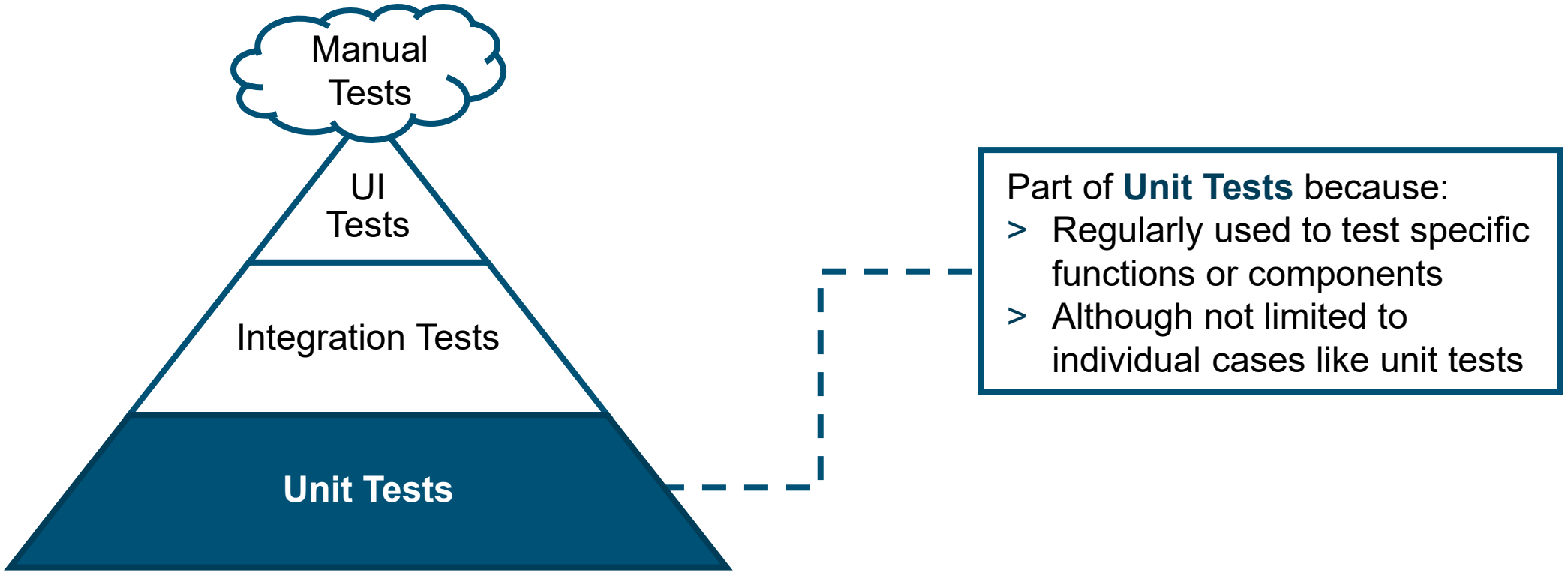
Parameterized Tests

Adaptable tests with variable inputs

Advantages and Disadvantages of Property-Based Testing

Advantages	Disadvantages
> Testing is less mundane > Leaving out important scenarios less probable due to randomness > Finding bugs nobody thought about > Increased flexibility because no need to (re-)write cases	> Finding specific bugs is difficult due to randomness > Reduced efficiency due to large amount of test cases executed > Writing specific data generators can be tedious, difficult and errorprone

Classification within the Test Pyramid



Based on: Vocke, H. (2018). The Practical Test Pyramid (2018-02-26) [Website]. https://martinfowler.com/articles/practical-test-pyramid.html

4

PROTOTYPE OF DATA ANALYSIS SCRIPTS USING HYPOTHESIS

How to Use Hypothesis

Tested in our environment with:

> Ubuntu 22.04.3 LTS
> Python 3.10.12
> pip 22.0.2

To install Hypothesis and Pytest simply run:

```
pip install hypothesis pytest
```

Basic Test

The **given** decorator is used to specify the inputs to the function you would like to test

```
1. from hypothesis import given
2. import hypothesis.strategies as st
3.
4. @given(st.integers())
5. def test_builtin_abs(x: int) -> None:
6.     assert abs(x) >= 0
7.     assert abs(x) == (x if x >= 0 else -x)
8.
9. test_builtin_abs()
```

Complex Input Generation

The **composite** decorator is used to combine input test generation methods (search strategies) into a single, more powerful and complex version

```
1.  from hypothesis.strategies import composite
2.
3.  PI = 3.14159
4.
5.  @composite
6.  def custom_input_generator(draw) -> tuple[float, str]:
7.      decimal = draw(st.floats(max_value=PI))
8.      text = draw(st.text(alphabet=st.characters(whitelist_categories=['Lu']), min_size=2, max_size=5))
9.      return decimal, text
10.
11. @given(custom_input_generator())
12. def test_custom_input_generator(generated_input: tuple[float, str]) -> None:
13.     decimal, text = generated_input
14.     assert decimal <= PI
15.     assert len(text) >= 2 and len(text) <= 5
16.     assert text.isupper()
17.
18. test_custom_input_generator()
```

Integration with Python

Integrating Hypothesis with Pytest is extremely easy, simply run:

```
pytest
```

As you would normally do. Either in the correct directory or by explicitly naming the file.

Testing a Data Analysis Script using Hypothesis: Calculate the Age of Astronauts

```python
import pandas as pd
from datetime import date, timedelta
from hypothesis import given, strategies as st
from pandas import DataFrame

def calculate_age(born):
    today = date.today()
    return today.year - born.year - ((today.month, today.day) < (born.month, born.day))

@given(st.dates(min_value=date(1920, 1, 1), max_value=date.today()))
def test_calculate_age(born: date) -> None:
    age = calculate_age(born)
    assert age >= 0 and age <= (born.today().year - born.year)

test_calculate_age()
```

Source: Stoffers, M., & Schlauch, T. (2021). Astronaut Analysis (2021-03-17) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.5018166

Discovering Issues with the help of Hypothesis: Example of an OverflowError

```
1.  def prepare_data_set(df):
2.      df = rename_columns(df)
3.      df = df.set_index("astronaut_id")
4.
5.      # Set pandas dtypes for columns with date or time
6.      df = df.dropna(subset=["time_in_space"])
7.      df["time_in_space"] = df["time_in_space"].astype(int)
8.      df["time_in_space"] = pd.to_timedelta(df["time_in_space"], unit="m")
9.      df["birthdate"] = pd.to_datetime(df["birthdate"])
10.     df["date_of_death"] = pd.to_datetime(df["date_of_death"])
11.     df.sort_values("birthdate", inplace=True)
12.
13.     # Calculate extra columns from the original data
14.     df["time_in_space_D"] = df["time_in_space"] / pd.Timedelta(days=1)
15.     df["alive"] = df["date_of_death"].apply(is_alive)
16.     df["age"] = df["birthdate"].apply(calculate_age)
17.     df["died_with_age"] = df.apply(died_with_age, axis=1)
18.     return df

        [...]
```

Source: Stoffers, M., & Schlauch, T. (2021). Astronaut Analysis (2021-03-17) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.5018166

Discovering Issues with the help of Hypothesis: Example of an OverflowError

```
@st.composite
def astronaut_data(draw):
    astronaut = draw(st.from_regex(r"http://www\.wikidata\.org/entity/Q\d+", fullmatch=True))
    astronautLabel = draw(st.from_regex(r"[A-Z][a-z]+ [A-Z][a-z]+", fullmatch=True))
    birthdate = draw(st.dates(min_value=date(1920, 1, 1), max_value=date(2030, 12, 31)) )
    birthplaceLabel = draw(st.from_regex(r"[A-Z][a-z]+", fullmatch=True))
    sex_or_genderLabel = draw(st.sampled_from(["male", "female"]))
    time_in_space = draw(st.integers(min_value=1))
    date_of_death = draw(st.one_of(
        st.none(),
        st.dates(birthdate + timedelta(days=1),
                 max_value=date(2030, 12, 31))
        )
    )

    [...]

test_prepare_data_set()
```

Source: Stoffers, M., & Schlauch, T. (2021). Astronaut Analysis (2021-03-17) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.5018166

Discovering Issues with the help of Hypothesis: Example of an OverflowError

```
> OverflowError: Python int too large to convert to C long
> Falsifying example: test_prepare_data_set(
>     data=[{'astronaut': 'http://www.wikidata.org/entity/Q0',
>       'astronautLabel': 'Aa Aa',
>       'birthdate': '2000-01-01T00:00:00Z',
>       'birthplaceLabel': 'Aa',
>       'sex_or_genderLabel': 'male',
>       'time_in_space': 18446744073709551616,
>       'date_of_death': '2000-01-02T00:00:00Z'}],
> )
```

Discovering Issues with the help of Hypothesis: Example of an OverflowError

```
1.  def prepare_data_set(df):
2.      df = rename_columns(df)
3.      df = df.set_index("astronaut_id")
4.
5.      # Set pandas dtypes for columns with date or time
6.      df = df.dropna(subset=["time_in_space"])
7.      df["time_in_space"] = df["time_in_space"].astype(int) # This line caused the error
8.      df["time_in_space"] = pd.to_timedelta(df["time_in_space"], unit="m")
9.      df["birthdate"] = pd.to_datetime(df["birthdate"])
10.     df["date_of_death"] = pd.to_datetime(df["date_of_death"])
11.     df.sort_values("birthdate", inplace=True)
12.
13.     # Calculate extra columns from the original data
14.     df["time_in_space_D"] = df["time_in_space"] / pd.Timedelta(days=1)
15.     df["alive"] = df["date_of_death"].apply(is_alive)
16.     df["age"] = df["birthdate"].apply(calculate_age)
17.     df["died_with_age"] = df.apply(died_with_age, axis=1)
18.     return df

        [...]
```

Source: Stoffers, M., & Schlauch, T. (2021). Astronaut Analysis (2021-03-17) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.5018166

5

PERSONAL REFLECTION

Property-Based Testing at the German Aerospace Center (DLR)

Relevance

Likely to be of interest for the data science researchers at DLR

Formal Verification

Uses wide input ranges to approximate formal verification

Simplicity

Easier compared to traditional unit testing methods

Beyond the Scope of German Aerospace Center (DLR)

Testing Flexibility

Increases adaptability and scope in software testing practices

Reusability

Improves efficiency through the reusability of test cases

Complements Unit Testing

Supportive method rather than replacement for unit testing

QUESTIONS AND ANSWERS