Utrecht University

May 29, 2024

Testing iRODS-based applications: Experiences with Yoda

Sirjan Kaur
s.kaur@uu.nl

Facts & Figures

PROFESSORS

>700

FACULTIES

7+2

teaching institutes

STUDENTS

>39,000

>25,000 Bachelor
>12,000 Master
>2,510 International
118 Nationalities

Utrecht University

Strategic Themes

- Strong links between teaching & research
- Integrated approach to academic / scientific and societal issues
- Strong disciplinary research programmes
- Eleven interdisciplinary focus areas

PATHWAYS TO SUSTAINABILITY

DYNAMICS OF YOUTH

INSTITUTIONS FOR OPEN SOCIETIES

LIFE SCIENCES

Utrecht University

What is Yoda?

- System to preserve, share, archive and publish research data during several stages of the research process
- Integrated Research Data Management solution
- Important strategic development for reaching Research Data Management goals

Yoda
research data management

Utrecht University

History

- Institutional service, developed and maintained by Utrecht University
- Cross domain archive and repository
- Developed as open-source software
- Sustained funding by the board, delivered through university corporate services

- Production service as of 2015
- First data package published in 2017
- Presented at the 2018 iRODS User Group Meeting
  - https://irods.org/uploads/2018/irods_ugm2018_proceedings.pdf
- Updates on Yoda presented at the 2023 iRODS User Group Meeting
  - https://irods.org/uploads/2023/irods_ugm2023_proceedings.pdf

Your research data and work according to FAIR principles, organize

Utrecht University

Yoda Consortium

Launched March 2023

Safeguard the development of Yoda as a national RDM platform

Effective collaboration on:
- Knowledge base
- Support for researchers
- Product development

SURF

Utrecht University

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

WAGENINGEN UNIVERSITY & RESEARCH

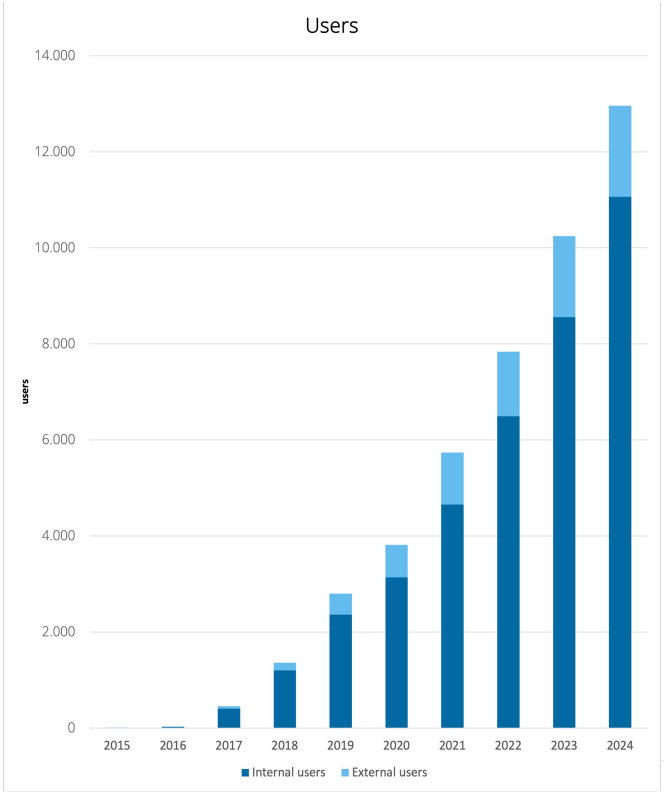
Amsterdam UMC

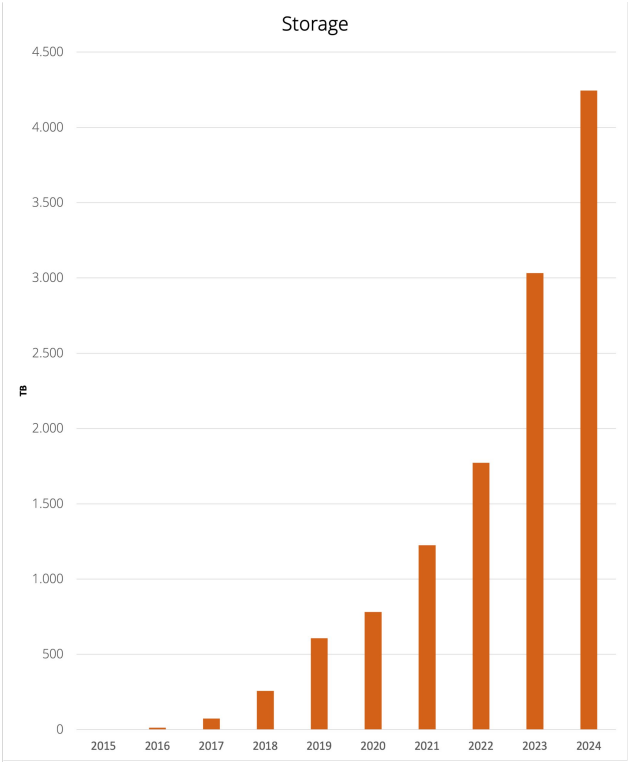
Erasmus Universiteit Rotterdam

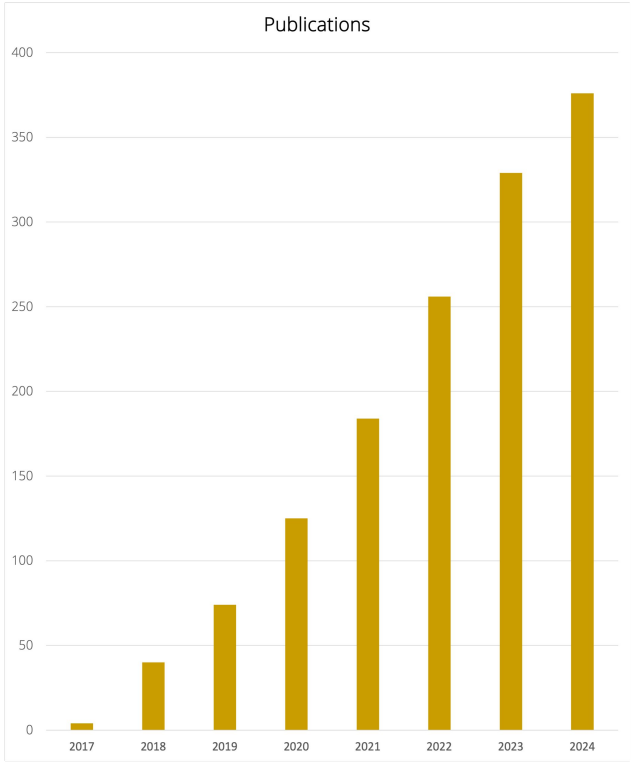
VU VRIJE UNIVERSITEIT AMSTERDAM

Universiteit Leiden

Utrecht University iRODS managed research data

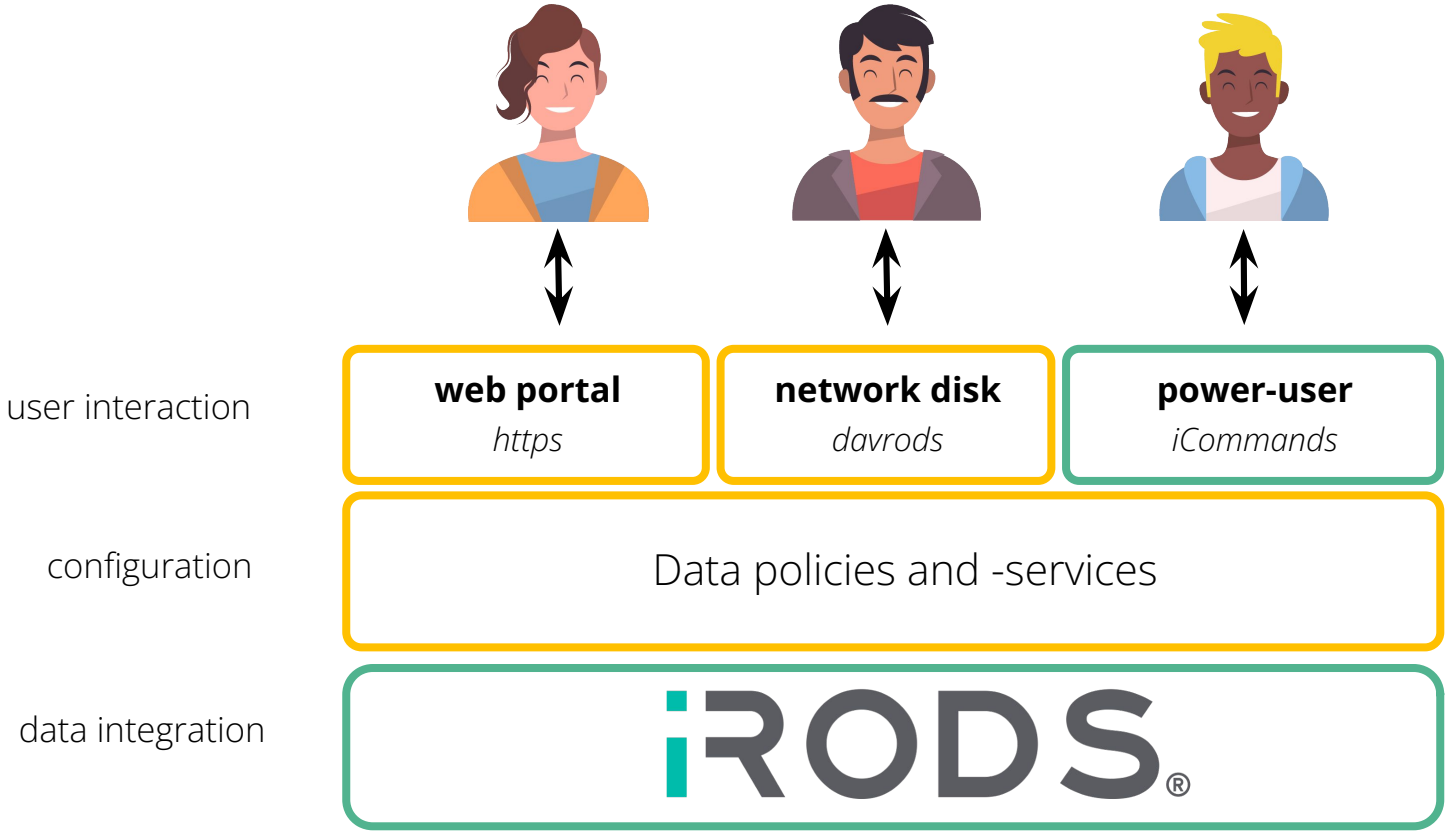




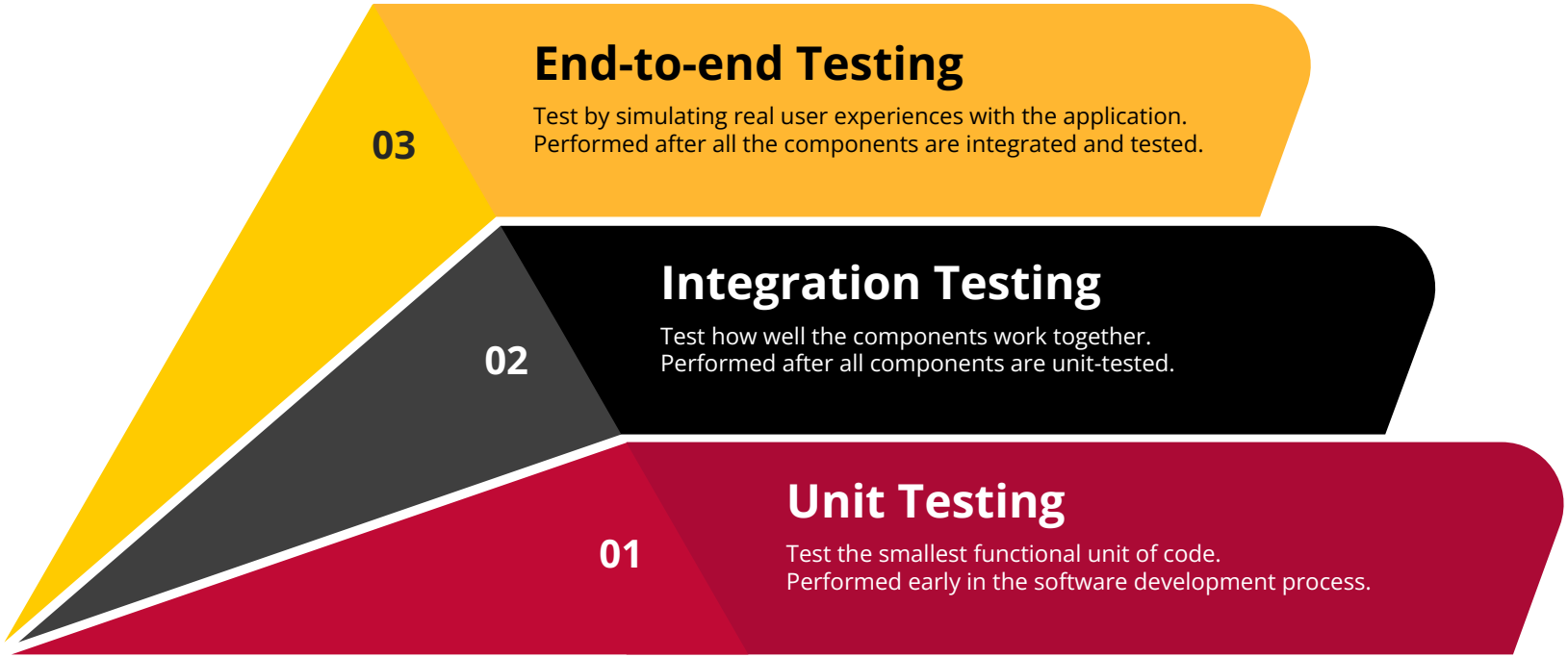


Utrecht University

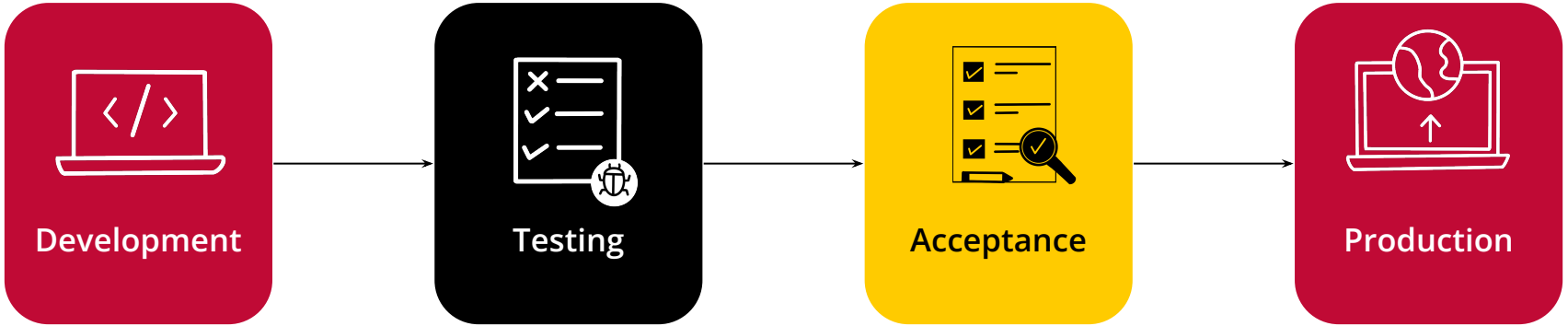
Yoda is built on iRODS



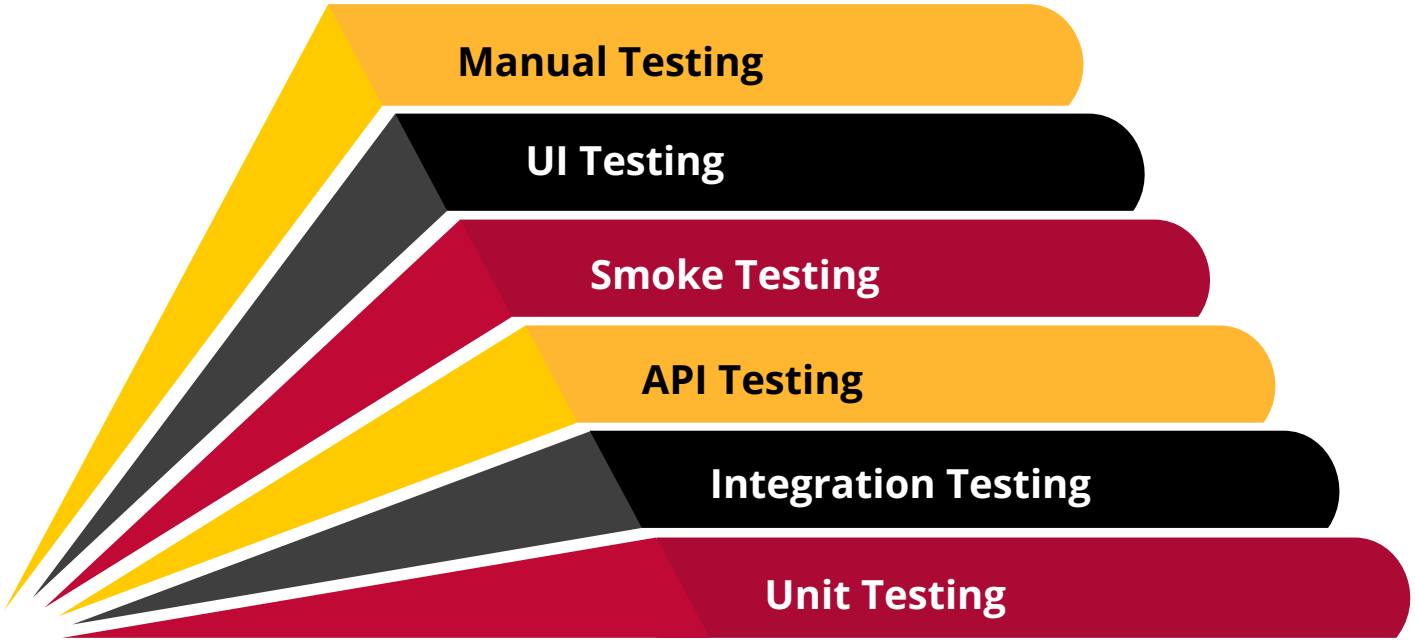
General Software Testing Strategy



Testing Strategy for Yoda



Testing Strategy for Yoda



Se

pytest

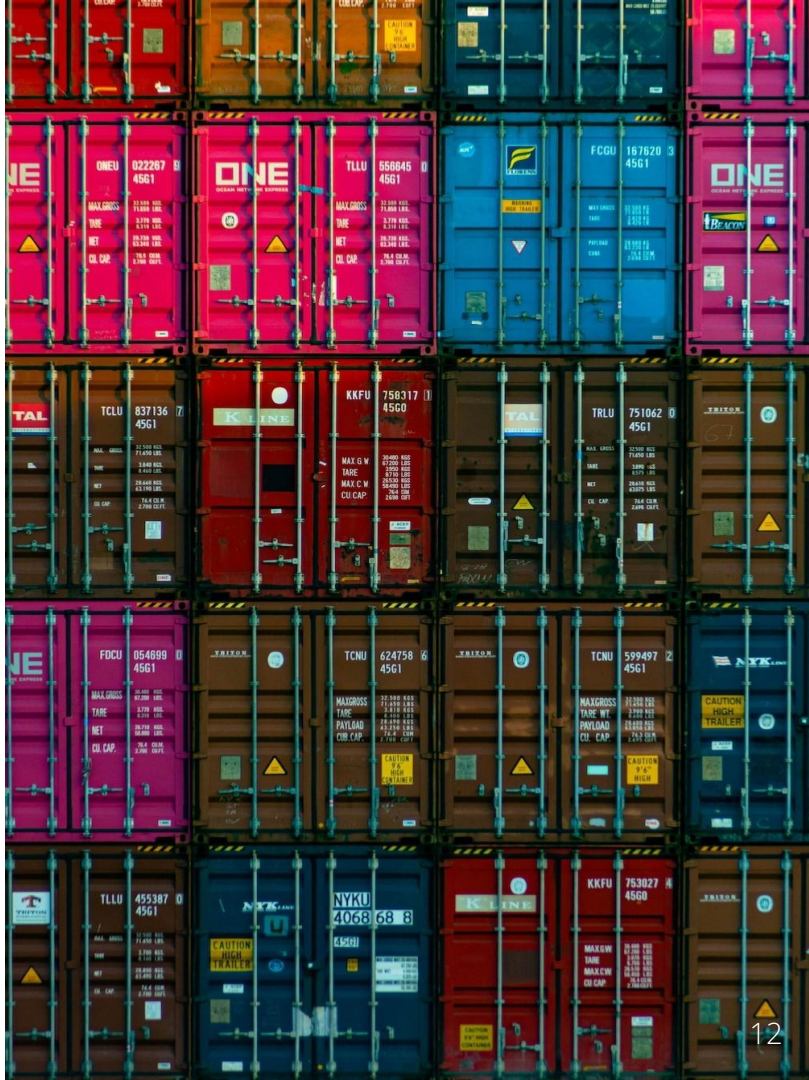
Docker

- Containerized version of Yoda
- Developed for local development
- Continuous integration

```
$ git clone https://github.com/UtrechtUniversity/yoda.git
$ cd yoda/docker/compose
$ docker compose pull
$ ../up.sh
```

More information:

https://utrechtuniversity.github.io/yoda/development/docker-setup.html



Unit Testing

- Test the smallest unit of code.
- Automated and scripted tests.
- Use python unittest framework for Yoda portal and MOAI and pytest for EUS.
- Run locally.
- Also run as part of GitHub workflow.
- Test suites differ per Yoda component.

More information:
https://utrechtuniversity.github.io/yoda/development/running-unit-integration-tests.html

```
# test_util_names.py
from unittest import TestCase

class UtilNamesTest(TestCase):

    def test_is_valid_category(self):
        self.assertEquals(is_valid_category(""), False)
        self.assertEquals(is_valid_category("foo"), True)
        self.assertEquals(is_valid_category("foo123"), True)
        self.assertEquals(is_valid_category("foo-bar"), True)
        self.assertEquals(is_valid_category("foo_bar"), True)

# unit_tests.py
from test_util_names import UtilNamesTest

def suite():
    test_suite = TestSuite()
    test_suite.addTest(...)
    test_suite.addTest(...)
    test_suite.addTest(makeSuite(UtilNamesTest))
    return test_suite

# Run unit tests
$ cd unit-tests
$ python -m unittest unit_tests
.................
----------------------------------------------------------------------
Ran 17 tests in 0.001s
```

Utrecht University

Integration Testing

- Test multiple components as a group.
- Automated and scripted tests.
- Use python unittest framework for MOAI and pytest for EUS. Tests can be run through integration rule in Yoda ruleset.
- Run either on development VM or dockerized version of Yoda using GitHub actions.

More information:
https://utrechtuniversity.github.io/yoda/development/running-unit-integration-tests.html

```
# integration_tests.py
__all__ = ['rule_run_integration_tests']

basic_integration_tests = [
    {"name":   "util.user.is_admin.yes",
     "test": lambda ctx: user.is_admin(ctx, "rods"),
     "check": lambda x: x},
    {"name":   "util.user.is_admin.no",
     "test": lambda ctx: user.is_admin(ctx, "researcher"),
     "check": lambda x: not x}
]

@rule.make(...)
def rule_run_integration_tests(ctx):
    ...


# run-integration-tests.r
def main(rule_args, callback, rei):
    result = callback.rule_run_integration_tests("")
    callback.writeLine("stdout", result["arguments"][0])

...

# Run integration tests
$ /usr/bin/irule -r irods_rule_engine_plugin-python-instance
    -F /etc/irods/yoda-ruleset/tools/run-integration-tests.r
util.user.is_admin.yes VERDICT_OK
util.user.is_admin.no VERDICT_OK
```

Utrecht University

API Testing

- Automated and scripted tests.
- Written with pytest-BDD.
- Run either on development VM or dockerized version of Yoda using GitHub actions.

More information:

https://utrechtuniversity.github.io/yoda/development/running-api-ui-tests.html

```
#api_browse.feature
@api
Feature: Browse API

    Scenario Outline: Browse folder
        Given user <user> is authenticated
        And the Yoda browse folder API is queried with <collection>
        Then the response status code is "200"
        And the browse result contains <result>

        Examples:
            | user       | collection                 | result     |
            | researcher | ~/research-initial         | testdata   |
            | researcher | ~/research-initial/testdata | lorem.txt  |


# test_api_browse.py
@given(parsers.parse("the Yoda browse folder API is queried with {collection}"),
    target_fixture="api_response")
def api_browse_folder(user, collection):
    return api_request(user, "browse_folder", {"coll": collection})

@then(parsers.parse("the browse result contains {result}"))
def api_response_contains(api_response, result):
    _, body = api_response

    assert len(body['data']['items']) > 0

    # Check if expected result is in browse results.
    found = False
    for item in body['data']['items']:
        if item["name"] == result:
            found = True

    assert found
```

UI Testing

- Automated and scripted tests.
- Written with pytest-BDD.
- Use pytest-splinter (Selenium wrapper) to automate browser actions.
- Tests are only run locally.

More information:

https://utrechtuniversity.github.io/yoda/development/running-api-ui-tests.html

```
#ui_login.feature
@ui
Feature: Login UI

    Scenario Outline: Invalid user login flow
        Given user is not logged in
        And the user is at the login gate
        When user <user> enters email address
        And user <user> logs in
        Then incorrect username / password message is shown

        Examples:
        | user                |
        | chewbacca@yoda.test |

# test_ui_login.py
@given('user is not logged in')
def ui_logout(browser):
    browser.visit(url) #url = "~/user/logout"

@given('the user is at the login gate')
def ui_gate(browser):
    browser.visit(url) #url = "~/user/gate"

@when(parsers.parse("user {user} enters email address"))
def ui_gate_username(browser, user):
    # Fill in username and click the 'Next' button

@when(parsers.parse('user {user} logs in'))
def ui_login(browser, user):
    browser.visit(url)
    # Fill in username and password and click the 'Sign in' button

@then("incorrect username / password message is shown")
def ui_user_incorrect(browser):
    assert browser.is_text_present("Username/password was incorrect")
```

Smoke and Manual Testing

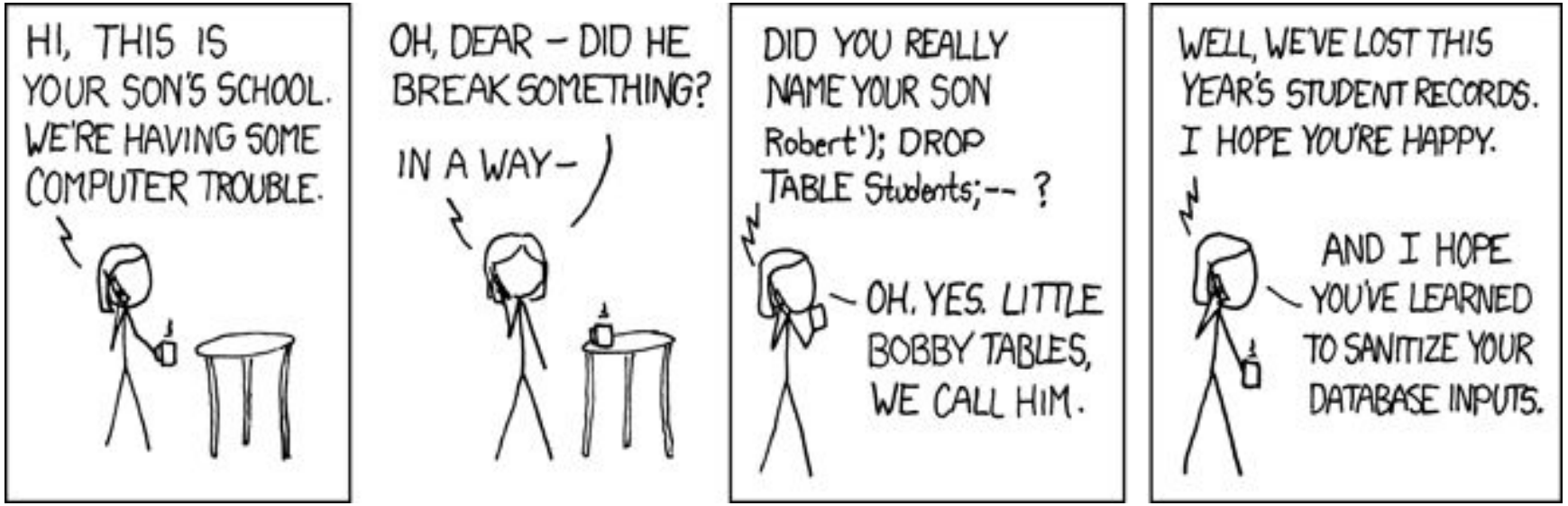
- Smoke tests are subset of tests to confirm that a Yoda environment has been deployed or updated correctly.
- Smoke tests are usually run in acceptance environment.
- Manual tests are run before a new release of software mainly for components that do not have good automated test coverage yet.
- Since much of our automated tests are run in development environment, manual tests ensure that the code works properly in acceptance and production environments.
- Maintain list of required functionality that needs to be tested manually.

Sample test set for Research module

ID	Ref	Prio	WAT testen (requirement, logisch testgeval)
1			**Research**
1.1			Research
1.1.1			*Browsing in folder*
1.1.2			*Show adjust number of folders*
1.1.3			*Show system metadata*
1.1.4			*Show provenance information*
1.1.5			*Show checksum report*
1.1.6			*Create folder*
1.1.7			*Upload small file (58kb)*
1.1.8			*Upload medium file > 330 MB*
1.1.9			*Upload large file (1gb)*
1.1.10			*Upload files*
1.1.11			*Upload folders*
1.1.12			*Play video (webm)*
1.1.13			*Check image (jpg)*
1.1.14			*Check for compliance with policy*
1.1.15			*Go to vault button*
1.1.16			*Lock folder/unlock folder*
1.1.17			*Lock found button*
1.1.18			*Submit folder to vault happy flow*
1.1.19			*Submit/unsubmit*
1.1.20			*Submit > reject*
1.1.21			*Submit > accept*
1.1.22			*Try to submit folder without metadata*
1.1.23			*Clean up temporary files*
1.1.24			*Upload files*
1.1.25			*Upload folders*
1.1.26			*File/folder Rename*
1.1.27			*File/folder Copy*
1.1.28			*File/folder move*

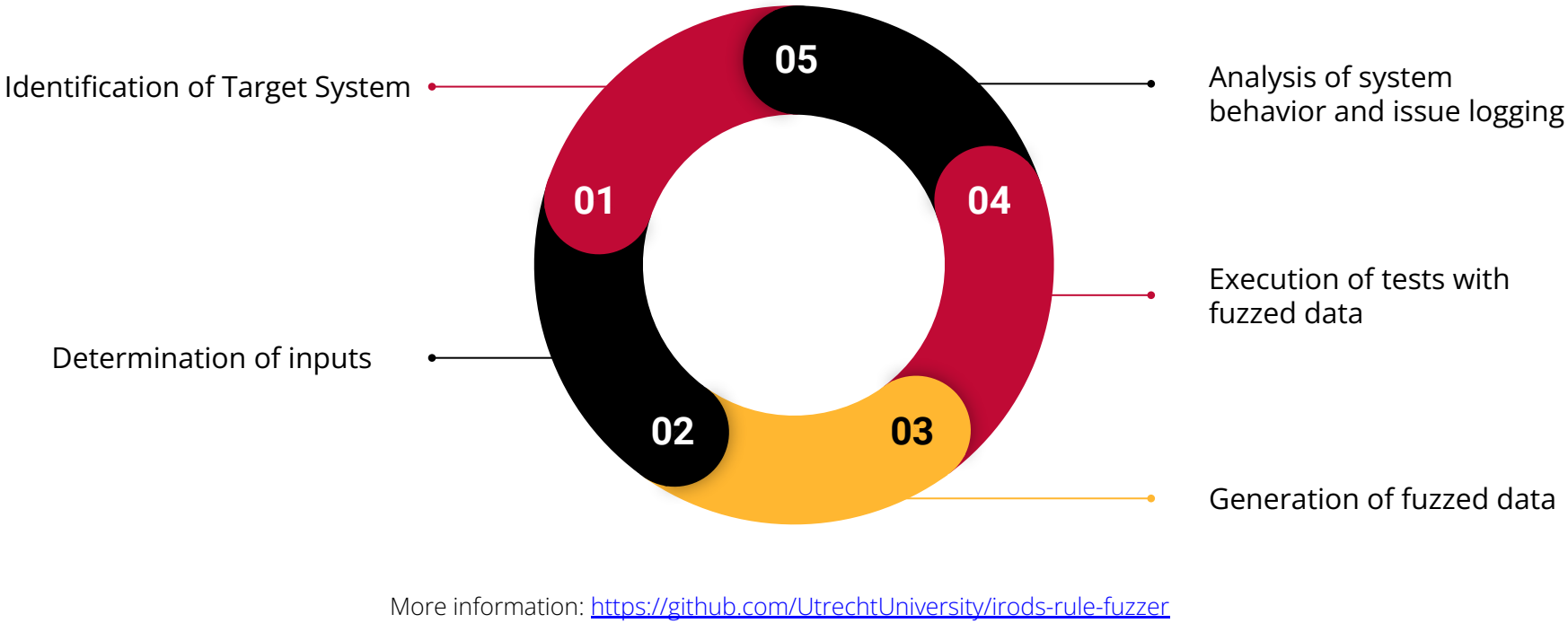
Utrecht University

Fuzzing and scriptless testing: testing the unexpected



https://xkcd.com/327 (CC-BY-NC)

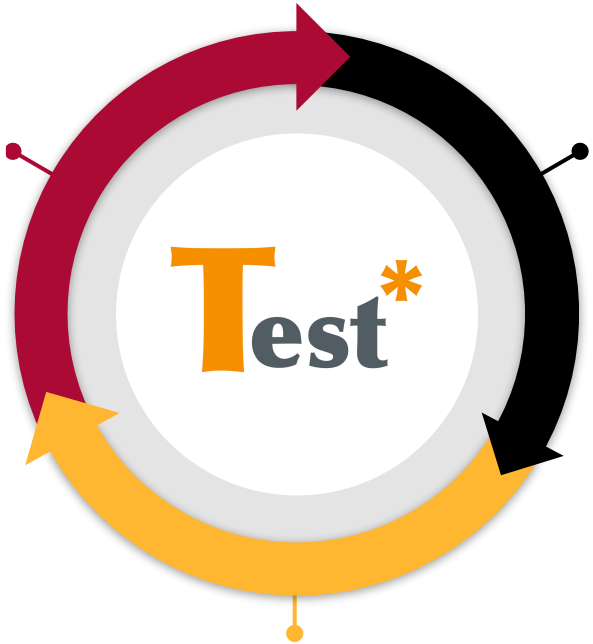
Fuzz testing



More information: https://github.com/UtrechtUniversity/irods-rule-fuzzer

Utrecht University

Scriptless GUI testing



Lessons learned

- Write rule preferably in Python, rather than in legacy rule language, to make it easier to test code automatically.
- Prioritize writing unit and low-level integration test over API and UI tests.
- Extract domain logic from the code that performs iRODS operations into separate functions or use dependency inversion principle to improve testability.

Future work

- Test coverage
  - Enhance the test coverage to include components that have limited or no automation.
- Reliable UI tests
  - Flaky in some systems, hence, cannot include them in Continuous Integration.
- Accessibility testing
  - Explore ways to incorporate this into our strategy.

$ iexit

Utrecht University

Sharing science,
shaping tomorrow