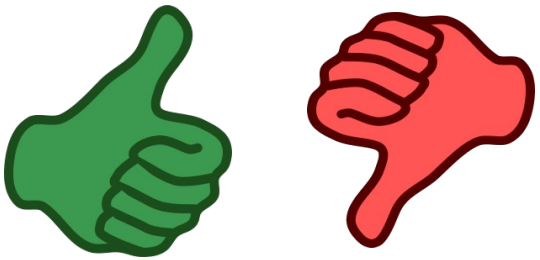
Robot Framework

Dos and Don'ts



Introduction

- Robot Framework is a generic acceptance level test automation framework
- This presentation demonstrates general guidelines how to create good test cases using it
- Both good practices and anti-patterns are presented
- See also http://code.google.com/p/robotframework/wiki/HowToWriteGoodTestCases

Most important goals

- Easy to understand
- Easy to maintain
- Fast to execute

Naming

- Very important to name well test cases, test suites, keywords, variables, libraries, resource files, ...
- Good name is explicit and easy to understand
- Consistency
- Namespaces
- Name should tell “what”, not “how”

Documentation

- Test suites often benefit from documentation explaining background etc.
- Well named tests created using well named keywords should not need extra documentation
- Reusable keywords must be documented
  - Good keyword and argument names help and are often adequate with higher-level keywords
  - Library keywords need detailed documentation

Example categorization

- **DO:** A very good and generally recommended practice.
- **DON'T:** An anti-pattern, should not be used.
- **TRY:** A practice that works in many contexts.
- **AVOID:** A practice that typically should be avoided but may sometimes be necessary.

DO: Appropriate abstraction level

```
*** Test Cases ***
Valid login
    Open browser to login page
    Input user name     demo
    Input password     mode
    Submit credentials
    Welcome page should be open
```

DON'T: Too low abstraction level

```
*** Test Cases ***
Valid login
    Open browser    ${LOGIN URL}    ${BROWSER}
    Set Selenium speed    ${DELAY}
    Maximize browser window
    Title should be    Login Page
    Input text    username_field    demo
    Input text    password_field    mode
    Click button    login_button
    Title should be    Welcome Page
    Location should be    ${WELCOME URL}
```

DON'T: Comments instead of abstraction

```
*** Test Cases ***
Valid login
    # Open browser to login page
    Open browser    ${LOGIN URL}    ${BROWSER}
    Set Selenium speed    ${DELAY}
    Maximize browser window
    Title should be    Login Page
    # Input user name
    Input text    username_field    demo
    # Input password
    Input text    password_field    mode
    # Submit credentials
    Click button    login_button
    # Welcome page should be open
    Title should be    Welcome Page
    Location should be    ${WELCOME URL}
```

DON'T: Documentation instead of abstraction

```
*** Test Cases ***
Valid login
    [Documentation]  Test for valid login:
    ...  - Open browser to login page
    ...  - Input user name
    ...  - Input password
    ...  - Submit credentials
    ...  - Welcome page should be open
    Open browser    ${LOGIN URL}    ${BROWSER}
    Set Selenium speed    ${DELAY}
    Maximize browser window
    Title should be    Login Page
    Input text    username_field    demo
    Input text    password_field    mode
    Click button    login_button
    Title should be    Welcome Page
    Location should be    ${WELCOME URL}
```

DO: Use data-driven style to avoid repeating workflows and to represent business rules

```
*** Settings ***
Test Template    Login with invalid credentials should fail

*** Test Cases ***      USER NAME          PASSWORD
Invalid user name       invalid            ${VALID PASSWORD}
Invalid password        ${VALID USER}      invalid
Both invalid            invalid            whatever
Empty user name         ${EMPTY}           ${VALID PASSWORD}
Empty password          ${VALID USER}      ${EMPTY}
Both empty              ${EMPTY}           ${EMPTY}
```

TRY: Gherkin style (ATDD/BDD)

```
*** Test Cases ***
Valid login
    Given login page is open
    When user "demo" logs in with password "mode"
    Then welcome page should be open
```

DON'T: Tests without checks

```
*** Test Cases ***
Valid login
    Open browser to login page
    Input user name    demo
    Input password    mode
    Submit credentials
```

AVOID: Tests with unrelated checks

```
*** Test Cases ***
Change password
    Open browser to login page
    Input user name    demo
    Input password    mode
    Submit credentials
    Welcome page should be open
    Open user management
    Select user    demo
    Change password    mode    newpwd
    Password changed successfully message should be shown
    Log out
    Input user name    demo
    Input password    newpwd
    Submit credentials
    Welcome page should be open
```

AVOID: Dependencies between tests

```
*** Test Cases ***
Valid login
    Open browser to login page
    Input user name    demo
    Input password    mode
    Submit credentials
    Welcome page should be open

Change password
    Open user management
    Select user    demo
    Change password    mode    newpwd
    Password changed successfully message should be shown

Login with new password
    Log out
    Input user name    demo
    Input password    newpwd
    Submit credentials
    Welcome page should be open
```

DO: Use teardowns for cleanup

```
*** Test Cases ***
Valid login
    Open browser to login page
    Input user name    demo
    Input password    mode
    Submit credentials
    Welcome page should be open
    [Teardown]    Close browser
```

DO: Use suite setup/teardown to speed up execution

```
*** Settings ***
Suite Setup       Open browser to login page
Test Template     Login with invalid credentials should fail
Test Teardown     Go to login page
Suite Teardown    Close browser

*** Test Cases ***      USER NAME        PASSWORD
Invalid user name       invalid          ${VALID PASSWORD}
Invalid password        ${VALID USER}    invalid
Both invalid            invalid          whatever
Empty user name         ${EMPTY}         ${VALID PASSWORD}
Empty password          ${VALID USER}    ${EMPTY}
Both empty              ${EMPTY}         ${EMPTY}
```

TRY: Move suite setup/teardown to directory level initialization files

```
*** Settings ***
Documentation     Child suites contain tests for valid and invalid login.
Suite Setup       Open browser to login page
Suite Teardown    Close browser
```

DO: Use variables to avoid hard coding

```
# GOOD

*** Variables ***
${LOGIN URL}    http://localhost:7272/
${BROWSER}      Firefox
${DELAY}        0.1

*** Keywords ***
Open browser to login page
    Open browser    ${LOGIN URL}    ${BROWSER}
    Set Selenium speed    ${DELAY}

# BAD

*** Keywords ***
Open browser to login page
    Open browser    http://localhost:7272/    Firefox
    Set Selenium speed    0.1
```

DON'T: Overuse variables

```
# BAD

*** Variables ***
${USERNAME FIELD}    username_field

*** Keywords ***
Input username
    [Arguments]    ${username}
    Input text    ${USERNAME FIELD}    ${username}

# GOOD

*** Keywords ***
Input username
    [Arguments]    ${username}
    Input text    username_field    ${username}
```

AVOID: Variable assignment on test case level

```
# GOOD

*** Test Cases ***
Withdraw from account
    Withdraw from account    $50
    Withdraw should have succeeded

# OK(ish)

*** Test Cases ***
Withdraw from account
    ${status} =   Withdraw from account    $50
    Withdraw should have succeeded    ${status}
```

AVOID: Complex logic in test data

```
*** Keywords ***
Get Max Line Length
    [Arguments]    ${string}
    ${max} =    Set Variable    ${0}
    @{lines} =    Split To Lines    ${string}
    :: FOR    ${line}    IN    @{lines}
    \    ${length} =    Get Length    ${line}
    \    ${max} =    Set Variable If    ${length} > ${max}
    \    ...    ${length}    ${max}
    [Return]    ${max}
```

DO: Move logic to libraries when possible

```
def get_max_line_length(string):
    lines = string.splitlines()
    if not lines:
        return 0
    return max(len(line) for line in lines)
```

DO: Use polling to synchronize
DON'T: Use sleeping to synchronize

```
# GOOD

*** Keywords ***
Download results
    [Arguments]    ${path}
    Initiate results downloading    ${path}
    Wait until created    ${path}

# VERY BAD

*** Keywords ***
Download results
    [Arguments]    ${path}
    Initiate results downloading    ${path}
    Sleep    10 seconds
```