

Why Writing Tests Does Save Time in Development

Magnus Mengelbier
Managing Director

Goals

- ☐ Write more code
- ☐ Write more documentation
- ☐ Writing more code and documentation because it saves time in development

The Thing

SAS Macro

```
%macro thing();  
  
  %* some code here ... ;  
  
%mend;
```

R Function

```
thing <- function() {  
  
  # some code here ...  
  
}
```

Python

```
class thing:  
  
    # some code here ...
```

- Moderately complex
- Intelligent
- Depends on an external resource, e.g. database or service
- A user's tool
- ... and it will be used for GCP-related activities, i.e. validated

How I ...



- Code file(s)
- Validation programs and documents
 - Plan(s) and report(s)
 - Requirements
 - Test specifications
 - Protocols
 - Test programs

- Examples
- Development scenarios
 - One or more programs
 - Unstructured
 - Unorganized

The Issue

- A user reports an issue
 - User's scenario is valid
 - Error message included
- Issue can be replicated
- Error message is not a custom error, e.g. root cause is in the code
- Error is internal and associated with a transient/temporary/work data object
- Failing line of code is a simple derivation (!?)
- Backtracking ... values for derivation is from a prior merge
 - One set of values is from an input data set (provided by user)
 - One set of values is from external resource, e.g. the database or service (available)
- Default value set in the merge is incompatible with user's scenario



Writing Tests

■ Validation

- Demonstrate a requirement
- Document that a function, feature or method is fit for its intended purpose
- Regulatory requirement
- Completed after development

■ Test-driven Development

- Verify a code block works as expected
- Created during development
- No formal requirement
- Replaces development scenarios
- Verifies code updates to not break
- Continuous process

■ Tests

- Collection of scenarios
- Unit tests
- Integration tests
- User Acceptance Tests

■ A Test

- Represents a specific scenario or condition
- Can be run in any order
- Not dependent on preceding or other tests

■ Test Framework

- Utility to execute and capture results of tests in a structured way

Principles of Testable Code

```
if ( <expr 1> ) {  
    # ... expression 1 is valid  
} else if ( <expr 2> ) {  
    # ... expression 2 is valid  
} else if ( <expr 3> ) {  
    # ... expression 3 is valid  
} else if ( <expr 4> ) {  
    # ... expression 4 is valid  
} else if ( <expr 5> ) {  
    # ... expression 5 is valid  
  
    #  
    # test this block  
    #  
} else if ( <expr 6> ) {  
    # ... expression 6 is valid  
}
```

R

- Principles
 - 1st : Must reach the code to test
 - 2nd : Must be able to trigger a scenario
- Construct-able scenarios
- Systematic challenges
- Positive versus negative

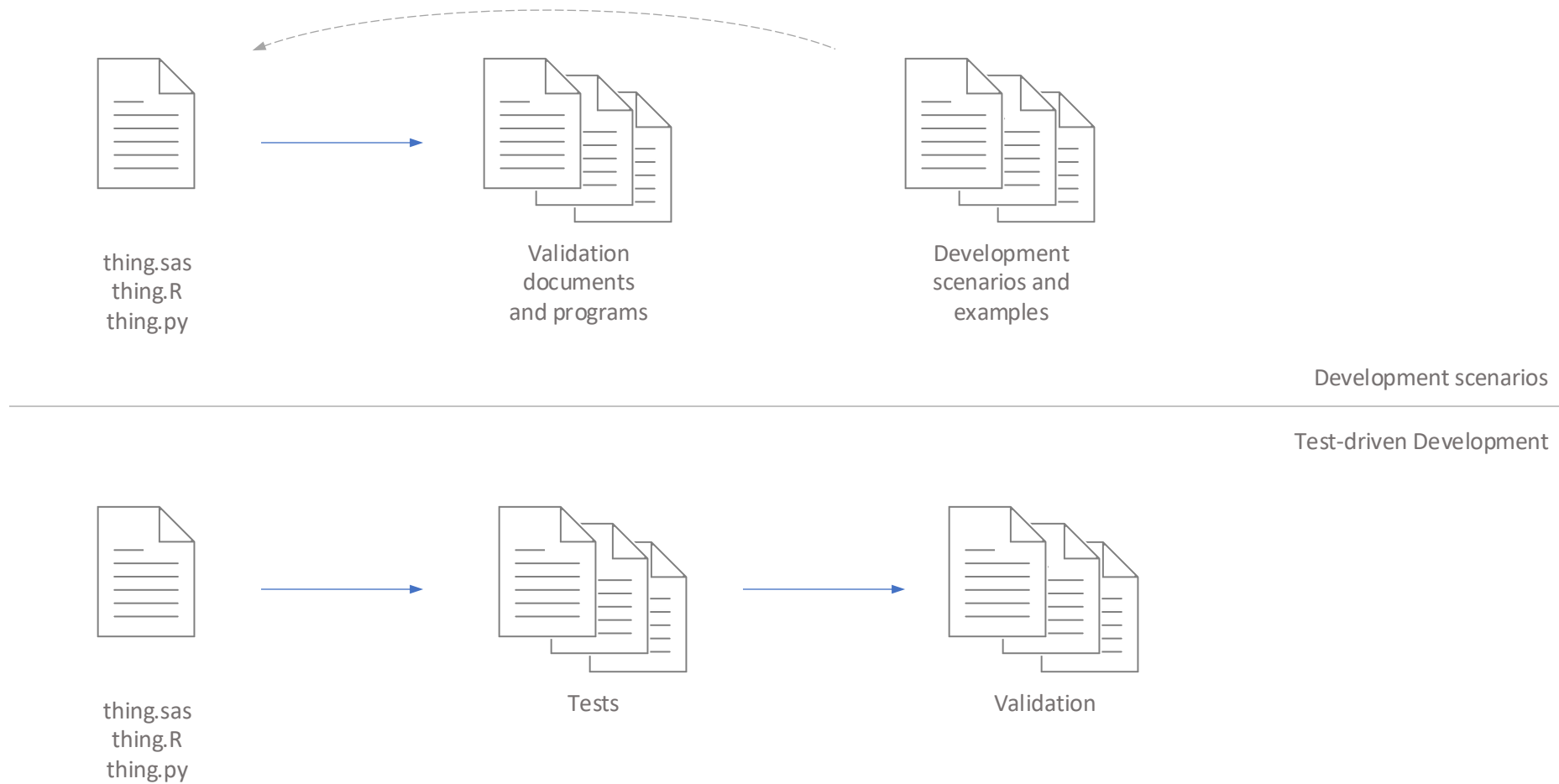
Writing Testable Code

```
if ( <expr 1> ) {  
    # ... expression 1 is valid  
} else if ( <expr 2> ) {  
    # ... expression 2 is valid  
} else if ( <expr 3> ) {  
    # ... expression 3 is valid  
} else if ( <expr 4> ) {  
    # ... expression 4 is valid  
} else if ( <expr 5> ) {  
    # ... expression 5 is valid  
  
    # trick: use an internal function  
    .my_internal_function( ... )  
  
} else if ( <expr 6> ) {  
    # ... expression 6 is valid  
}
```

R

- All code has a purpose
- During development
 - Consider how a code block can be tested
 - Annotate code, i.e. comments
 - Construct one or more test scenarios
 - Consider the validation requirement
- Use internal functions, methods and classes
 - Reachable
 - Testable
- Include the positive and the negative
- Execute tests often
- Consider automation
- Validation is a formality

Development effort



Déjà vu

- A user reports an issue
 - User's scenario is valid
 - Error message included
- Issue can be replicated
- Error message is not a custom error, e.g. root cause is in the code
- Error is internal and associated with a transient/temporary/work data object
- Failing line of code is a simple derivation (!?)
- Backtracking ... values for derivation is from a prior merge
 - One set of values is from an input data set (provided by user)
 - One set of values is from external resource, e.g. the database or service (available)
- Default value set in the merge is incompatible with user's scenario

In summary

- Test-driven development
- Promotes good coding practice
- Testable code
- Less duplication
- Validation is a formality

Magnus Mengelbier

magnus.mengelbier@limelogic.com

Limelogic AB

Jungmansgatan 12
SE-211 11 Malmö
Sweden

www.limelogic.com

orgs

github.com/limelogic
github.com/openapx
github.com/cxlib